ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Privacy-Preserving Machine Learning for Cyber Insurance

Master Thesis

T. Giovanna

February 23, 2021

Supervisor: Prof. Dr. S. Čapkun
Advisors: Prof. E. Mohammadi, Dr. K. Kostiainen

Department of Computer Science, ETH Zürich

Abstract

Cyber risk assessment is an emerging field, in particular for insurance companies, that offer cyber insurance packages to their clients. However, understanding cyber risk is not easy, especially due to the lack of historical data. While machine learning can help, reconciling accuracy, explainability and privacy when data is lacking is challenging. Some machine learning techniques are better than others at addressing these challenges. For instance, ensemble methods such as gradient boosted decision trees (GBDT) are easily explainable, due to their tree structure. This is not the case of other methods such as deep neural networks, which are much more complex to interpret. GBDT models can also provide privacy through differential privacy. However, current state of the arts on differentially private GBDT models suffers from low accuracy when there are limited training data. In this thesis, we propose a new decision tree induction algorithm, *2-nodes*, that enhances accuracy over small datasets while satisfying ε-differential privacy. Further, we propose an algorithm based on Bayesian networks to generate synthetic data for cyber risk assessment. Finally, we evaluate our model on various real and synthetic datasets and show that our new induction method is able to improve accuracy on small datasets.

Contents

Chapter 1

Introduction

1.1 Motivation

In many scenarios where machine learning (ML) is involved, the responsible developing team strives to have an accurate, explainable model. Accurate, because the end goal of an ML model is to help the team in understanding a problem better. Inaccurate outputs would not be helpful. Explainable, because if the team cannot understand the decisions made by the model, in particular the steps involved in these decisions, the problem would remain as complex as it was. Any machine learning model needs a starting point. This starting point usually takes the form of a *training set*.

The training set contains data about the problem to be learned about, and sometimes data about the solution to be found (*unsupervised* vs. *supervised* learning). In some cases, leakage of the training data creates serious privacy issues, which for some applications is unacceptable. This leads the model to not only require being accurate and explainable, but also to prevent such privacy leaks. One way to prevent these leaks is to train the model in a privacy preserving fashion. Privacy preserving machine learning models can offer guarantees that prevent information about the training data to be extracted. An example application where privacy of the training data is important can be found in assessing *cyber risk*, which we focus on in this thesis.

In recent years, cyber attacks and data breaches have surged, as outlined by recent reports ([40], [21], [9]). Companies, large and small, have an increasing need for protection. Such protection not only extends to their technical infrastructure and ability to respond to incidents, but also to potential financial losses that they may face. To cover the latter, insurance companies have come up with cyber insurance products. From the insurer perspective's however, assessing the cyber risk that a particular customer is exposed to is highly challenging. The understanding of cyber risk, i.e. how to describe or model the risk, is not yet as understood as other risks, such as the ones covered by car insurance products.

One major issue that insurers face is trustworthiness when it comes to data provided by their customers. Indeed, customers are often unwilling to disclose their true security practices, fearing that such information could be used to discriminate against them, should they not be complying with or implementing these practices according to the latest industry standards. Currently, these information are collected by insurers through questionnaires that the customers need to fill in. These questionnaires will typically include questions about their security management practices, e.g. details about their software patching process or remote access policy.

1.2 Example setting

One way to address above problems is for insurers to give customers access to an interface allowing them to answer their questionnaires, within a protected space (such as an Intel SGX enclave). This protected space:

- can be used to collect sensitive data, while reassuring both the insurers and the customers that no party has direct access to the data.
- can be leveraged by customers through remote attestation to verify the correctness of the protected space (i.e. the integrity of any code running within the enclave).
- can give customers privacy guarantees with respect to the data they provide.

This is represented on the left hand side of Figure 1.1. In this thesis, we focus on enabling a trustworthy evaluation of the data collected within the protected space. The goal of this evaluation is to train a machine learning model that can be used by insurers to help them evaluate the risk and potential losses associated to the onboarding of a new customer, while providing customers peace of mind about their data. This is represented on the right hand side of Figure 1.1.

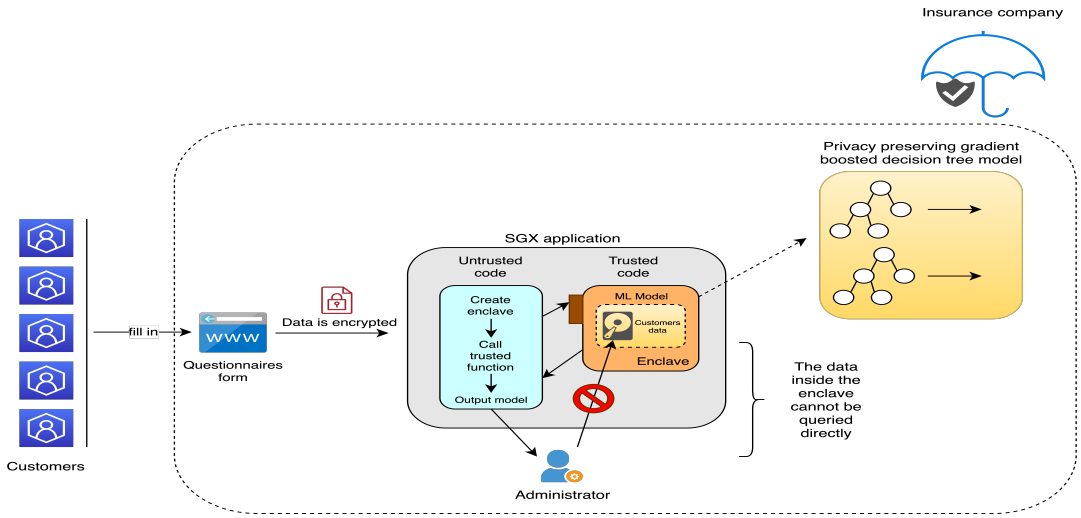


Figure 1.1: The SGX application on the left hand side, and the privacy preserving machine learning operating within the SGX enclave, zoomed in on the right hand side. SGX application schematic adapted from [20].

The insurance company hosts the enclave. The data that the customers send are sent encrypted to the enclave, and never leave it. The data cannot be queried directly, be it by the customers or the insurance company. The SGX enclave can then release the trained model to the insurance company, which can leverage it to learn various kinds of information about the customers. However, due to the privacy preserving properties of the model, the insurance company cannot retrieve information about any single customer.

1.3 Chosen approach

Gradient Boosted Decision Tree (GBDT) models have attracted a lot of attention in recent years and have successfully been used as a winning model in various machine learning competitions[1]. GBDT models have been shown to be performant, and easily explainable due to their

[1] https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions

tree structure. In addition, research work (e.g. of Qinbin et al. (2020) [25] or Liu et al. (2018) [26]) has pushed GBDT models towards becoming privacy preserving by leveraging differential privacy. While differential privacy is not the only way to provide privacy (see Chapter 8), its strong mathematical foundations and provable properties have made it a de-facto choice in many research work. Although our example setting justifies the attack model that we consider in Chapter 7, this thesis focuses on the improvement and evaluation of the algorithmic parts: differential privacy applied to gradient boosted decision trees.

1.4 Contributions

While recent work shows encouraging results, developing gradient boosted decision tree models that satisfy ε-differential privacy while remaining as accurate as non-private models is still an open challenge, especially when the training data is small, such as in the case of cyber risk evaluation. This thesis aims at addressing the current shortcomings of previous approachs, with the following contributions:

1. We propose a new decision tree induction method, called *2-nodes*, that enhances accuracy over low-populated datasets, while satisfying ε-differential privacy. In particular, we propose to make use of extra data in the tree induction process, by finding the optimal splitting point over a node and its sibling's data rather than just the node itself. We use this induction method in an implementation of DP-GBDT that we implement from the literature. This is covered in Chapter 4.
2. In Chapter 5, and since real data is lacking, we propose a way to generate synthetic datasets that mimic cyber insurance questionnaire answers. To achieve this, we collect figures from cyber security reports written by different vendors in the security and insurance sector, and derive a Bayesian network that we use to compute the dataset's features and targets.
3. We show in Chapter 6 that our model can successfully be used to accurately evaluate cyber risk on 4 different synthetic datasets, as well as improve predictions over low-populated datasets.
4. We explore privacy attacks on machine learning models in Chapter 7 and apply them to our DP-GBDT model, and show that in some cases it can reduce attack accuracy under strong privacy constraints.

Chapter 2

Background

2.1 Decision trees

A decision tree is a supervised learning technique, where given a set of inputs $\mathbf{X} = X_1, \ldots, X_n$ one tries to predict a response or class Y for an unseen input X_i. Decision trees can be used for both classification and regression tasks. In a classification task, Y is a class (also called a label): it is *categorical* (or *discrete*). Consider for instance a set of inputs $\mathbf{X}$ where each instance X_i ($i \in [1, n]$) describes the size and the colour of a fruit. For each instance, the associated class in $\mathbf{Y}$ could be the type of fruit. A decision tree that is fit on this dataset will learn how to characterise a new, unseen fruit X_{new}. In such a scenario, the classification task will therefore be to predict if X_{new} is, say, an apple or an orange. On the other hand, in a regression task, Y is a quantity, a number: it is *numerical* (or *continous*). If our instances in $\mathbf{X}$ were now to describe the type and the weight of the fruit, with associated class their size in centimetres, then the regression task would be to predict a real number for the size of X_{new}. Figure 2.1 shows a simple decision tree for both classification and regression.

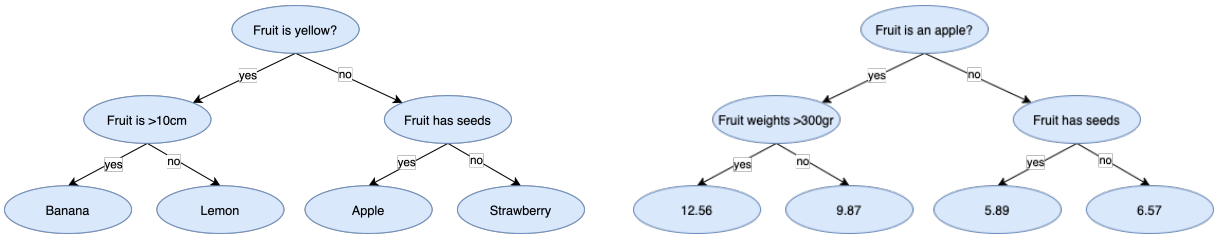


Figure 2.1: A classification tree (left) and a regression tree (right).

Decision trees consist of:

- Nodes: a node represents a decision (or split) to be made about its instances $X_i, \ldots, X_j$, where $i, j \in [1, n]$ and $i < j$. This split happens on an attribute and attribute's value of an instance. In the rightmost decision tree in Figure 2.1, one such split is on attribute *type of fruit* and attribute's value *apple*.
- Edges: an edge corresponds to the outcome of the node's decision, and connects to the next node.
- Leaf nodes: a leaf node is a node that is terminal, i.e. it has no children. Leaf nodes hold the prediction results.

To classify a new, unseen instance, one simply follows the tree edges until it reaches a leaf node. Constructing a decision tree usually involves a recursive algorithm, such as ID3 [33].

In ID3, the tree construction starts from the root node, and progresses towards the leaf nodes until a maximum depth is reached, controlling how many recursions the algorithm will run through. At each node, the splitting point (i.e., the attribute and attribute's value on which to separate the node's instances) is chosen using the *information gain*.

Definition 2.1 (Information Gain [3]) *Let $\boldsymbol{X} = X_1, \ldots, X_n$ be a set of inputs (training instances), where each X_i is of the form $(\boldsymbol{x}, y) = (x_1, \ldots, x_n, y)$. x_j is the value of the j^{th} attribute of $\boldsymbol{x}$, and y is its corresponding label. The information gain for an attribute j is given by:*

$$IG(\boldsymbol{X}, j) = H(\boldsymbol{X}) - H(\boldsymbol{X}|j)$$

Where $H(\cdot)$ is the Shannon entropy. In other words, the information gain is the difference in entropy before and after the potential split on attribute j. Note that after the split, both entropies of resulting nodes are taken into account and added together.

We can use this measure to rank attributes and choose the best splitting point. While decision trees are rather simple to understand, they come with their fair share of drawbacks:

- Prone to overfitting (the trees fit perfectly the training samples but fail to accurately predict unseen samples).
- Highly complex when there are many class labels / depth is large.
- Low bias and high variance (trees make no assumptions about the target variables, however a change in the training dataset may result in a completely different tree structure).

To address the above limitations, one can train multiple trees. In this case, we talk about *ensemble methods*. The idea behind ensemble methods is to combine multiple *weak learners* to form a *strong learner*. Here, a weak learner is a single decision tree and a strong learner is an ensemble of decision trees. The most common ensemble methods are *bagging* and *boosting*. In bagging, one trains several trees on different subsets of the data. As a result, the trees have a different structure and therefore offer different predictions. These trees can be trained in parallel, as they are independent from one another. The final predictions can be computed as the average of all predictions of each individual decision tree. A popular extension of bagging is a *Random Forest* model, where the subsets of data and the trees splitting points are chosen at random.

In boosting, the trees are fitted sequentially, meaning that for an ensemble of n trees, tree t depends on tree $t-1$ for all $t \in [1, n]$. In particular, tree t fits the residual of tree $t-1$. This iterative process is repeated n times, for all trees. Each tree's output ($h(x)$) is usually given a weight w relative to its accuracy. The final predictions are computed as the weighted sum of each tree: $\hat{y}(x) = \sum_t w_t h_t(\mathbf{x})$ [42]. The more often an instance is misclassified, the more important it becomes for the training of the subsequent tree. Training the trees therefore becomes a minimisation problem on the objective function $O(\mathbf{x}) = \sum_i l(\hat{y}_i, y_i) + \sum_t \Omega(f_t)$ [7] where $l(\hat{y}_i, y_i)$ is the loss function (i.e. the distance between the truth and the prediction of the i^{th} sample) and $\Omega(f_t)$ is the regularisation function (it penalises the complexity of the t^{th} tree). When the objective function is minimised using *gradient descent*, we talk about *gradient boosting*, which we introduce in the next section.

2.2 Gradient boosted decision trees (GBDT)

Given a convex loss function l and a dataset with n instances $\mathbf{X} = X_1, \ldots, X_n$, $X_i = (\mathbf{x}, y) = (x_1, \ldots, x_d, y)$ $\forall i \in [1, n]$, GBDT minimises the following objective function at the t^{th} iteration [38]:

$$O(\mathbf{x})^{(t)} = \sum_{i}^{n} \left(g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right) + \Omega(f_t) \tag{2.1}$$

where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ is the first order gradient statistics of the loss function, f_t is the decision tree at iteration t, and $\Omega(f_t) = \frac{1}{2}\lambda \|V\|^2$ is the regularisation term (V is the leaf weight, λ is the regularisation parameter). The tree is built from the root until maximum depth is reached. Assume that I_L and I_R are the instances in the left and right nodes after a split. We have $I = I_L \cup I_R$, and the gain after the split is given by: [7]

$$G(I_L, I_R) = \frac{1}{2} \left(\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right) \tag{2.2}$$

For the square loss function, $h_i = 1$ in Equations 2.1 and 2.2. Furthermore, since the last term in the above equation does not depend on the splitting point, and assuming the square loss function is used, Equation 2.2 can be further simplified to: [25]

$$G(I_L, I_R) = \frac{(\sum_{i \in I_L} g_i)^2}{|I_L| + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{|I_R| + \lambda} \tag{2.3}$$

GBDT traverses all combinations of features and features' values to find the best split. If a node has a negative gain or if it is at the maximum depth, then it becomes a leaf node and it is assigned the optimal leaf value:

$$V(I) = -\eta \frac{\sum_{i \in I} g_i}{|I| + \lambda} \tag{2.4}$$

where η is the learning rate, which controls the influence of a single tree. Algorithm 1 describes the training process of gradient boosted decision trees.

Algorithm 1: GBDTs training process

Input: $\mathbf{X} = X_1, \ldots, X_n$: instances, $\mathbf{y} = y_1, \ldots, y_n$: labels
Input: λ: regularisation parameter, d_{max}: maximum depth, η: learning rate
Input: T: total number of trees, l: loss function
Output: An ensemble of trained decision trees.

```
for t = 1 to T do
    Update gradients of all training instances on loss l
    for d = 1 to d_max do
        for each node in current depth do
            for each split value i do
                G_i ← (Σ_{i∈I_L} g_i)^2 / (|I_L|+λ) + (Σ_{i∈I_R} g_i)^2 / (|I_R|+λ)   ▷ Equation 2.3
            Split node on split value i = arg max_i(G_i)
    for each leaf node i do
        V_i ← −η Σ_{i∈I} g_i / (|I|+λ)                                                 ▷ Equation 2.4
```

2.3 Differential privacy

In some cases, the privacy of the individuals that constitute a dataset needs to be preserved. There are several ways to do that, such as *K-anonymity*. In *K*-anonymity, the attributes of the individuals are generalised such that a single individual becomes indistinguishable from at least $K-1$ other individuals. Unfortunately, achieving optimal *K*-anonymity is NP-hard and there has been criticism about possible re-identification of the individuals [18], in particular when the attacker acquires background information about the training data.

A different approach is to use *differential privacy*. Differential privacy is a mathematical definition that provides provable privacy guarantees [10]. It assumes that a potential attacker has almost full knowledge about the training data, and is only uncertain about a single training data point. By leveraging differential privacy, a defender can deny the presence of each and every data point in the training data.

Definition 2.2 (ε-Differential Privacy [25]) *Let ε be a positive real number and f be a randomised function. The function f is said to provide ε-differential privacy if, for any two datasets D and D' that differ by a single record and any output O of function f:*

$$Pr[f(D) \in O] \leq e^{\varepsilon} \cdot Pr[f(D') \in O] \tag{2.5}$$

In the above definition, ε is called the *privacy budget*. Intuitively, this budget measures how much privacy the function f offers. The lower this budget, the higher privacy f achieves, i.e. the more a defender can deny the presence of single data points in the training data. To achieve ε-differential privacy in practice, the Laplace mechanism and the exponential mechanism are usually adopted [11]. The process involves adding noise that is calibrated to the *sensitivity* of f.

Definition 2.3 (Sensitivity [25]) *Let $f : \mathcal{D} \to \mathcal{R}^d$ be a function. The sensitivity of f is:*

$$\Delta f = \max_{D,D' \in \mathcal{D}} \|f(D) - f(D')\|_1 \tag{2.6}$$

where D and D' have at most one different record.

The sensitivity of f captures the magnitude by which a single datapoint in $\mathcal{D}$ can change the function f in the worst case. In other words, it is a measure of the randomness that we must introduce in order to hide the participation of a single datapoint (to preserve the datapoint's privacy). The ε-deniability that a defender gets can fail with probability at most Δf. Usually, this randomness is drawn from a Laplace distribution.

Definition 2.4 (Laplace Distribution) *A random variable has a $Lap(\mu, b)$ distribution if its probability density function is:*

$$f(x|\mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right) \tag{2.7}$$

Where μ is the mean and b is the scale of the distribution.

Combining the previous definitions, we can formally describe the Laplace mechanism and the exponential mechanism.

Theorem 2.5 (Laplace Mechanism [25]) *Let $f : \mathcal{D} \to \mathcal{R}^d$ be a function. The Laplace mechanism F is defined as:*

$$F(D) = f(D) + Lap(0, \Delta f/\varepsilon) \tag{2.8}$$

Where the noise $Lap(0, \Delta f/\varepsilon)$ *is drawn from a Laplace distribution with mean* $\mu = 0$ *and scale* $b = \Delta f/\varepsilon$*. Then F provides ε-differential privacy.*

Theorem 2.6 (Exponential Mechanism [25]) *Let* $u : (\mathcal{D} \times \mathcal{R}) \to \mathbb{R}$ *be a utility function, with sensitivity* Δu*. The exponential mechanism F is defined as:*

$$F(D, u) = \textit{choose } r \in \mathcal{R} \textit{ with probability} \propto \exp\left(\frac{\varepsilon u(D, r)}{2\Delta u}\right) \tag{2.9}$$

Then F provides ε-differential privacy.

Theorems 2.5 and 2.6 provide privacy guarantees for a single function. When using multiple functions, much like in an algorithm, there are considerations to take into account around the privacy budget ε. This is considered through two additional theorems: *sequential* composition theorem and *parallel* composition theorem ([11], [25]).

Theorem 2.7 (Sequential Composition) *Let* $f = \{f_1, \ldots, f_m\}$ *be a series of functions performed sequentially on a dataset. If* f_i *provides* ε_i*-differential privacy, then f provides* $\sum_i^m \varepsilon_i$*-differential privacy.*

Theorem 2.8 (Parallel Composition) *Let* $f = \{f_1, \ldots, f_m\}$ *be a series of functions performed separately on disjoint subsets of a dataset. If* f_i *provides* ε_i*-differential privacy, then f provides* $\{\max(\varepsilon_1, \ldots, \varepsilon_m)\}$*-differential privacy.*

Chapter 3

Differentially Private Decision Trees

Differentially private decision tree learning methods aim at preventing information leakage at two levels: when splitting internal nodes, and when querying leaf nodes for prediction results. In [16], Fletcher et al. (2020) survey differential privacy methods for decision tree learning. They identify the main factors that need to be considered when designing such algorithms:

- **The privacy budget** ε.
- **The number of times the data needs to be queried**. This directly impacts budget consumption. If the data needs to be queried multiple times, then we must apply the sequential composition theorem (Theorem 2.7). It is clear from the theorem that the more queries there are, the faster the privacy budget will be consumed. The querying strategy of a differentially private model is therefore crucial.
- **The sensitivity** Δ **of the queries**. For a fixed privacy objective, tighter sensitivity bounds lead to less added noise, as shown in Theorem 2.5 and 2.6: in the Laplace mechanism, reducing the sensitivity Δf will reduce the scale of the associated Laplace distribution, hence decreasing the amount of added noise. Equally, in the exponential mechanism, reducing the sensitivity Δu will make sure that the chosen attribute's value has a high probability of being amongst the best possible choices.
- **The size of the dataset**. The larger the dataset is, the easier it is for the trees to learn the underlying data structure (since they have access to more data to make better splits), and the least sensitive to noise the trees will be (since a large amount of data will make sure that best splits are chosen with high probability).

3.1 Related work

In this section, we explore current and past related work.

3.1.1 Privacy budget allocation

Jagannathan et al. (2009) [22] are amongst the first ones to propose a practical differentially private implementation of decision trees for classification problems. They propose a differentially private version of the ID3 algorithm, and show that a naive implementation where counting queries are noised lead to very poor utility under realistic privacy budget constraints. In fact, splitting a privacy budget of $\varepsilon = 1$ evenly across all queries achieves a classification error rate as high as 81.27% on the nursery dataset[1]. To address this shortcoming, they propose a new approach based on *Random Forest* ensemble methods. Instead of adding noise to each

[1] https://archive.ics.uci.edu/ml/datasets/nursery

of the queries made during the ID3 algorithm, the decision trees are computed at random and noise is only added to the leaf nodes. This leads to tremendous improvements, with the previously reported error rate dropping to approximately 10.5% for the same dataset and under similar privacy constraints. For comparison, the non-private version of the ID3 algorithm achieves an error rate of 1.81% on the same dataset.

In [27], Mohammed et al. (2011) split the privacy budget ε into two halves, where each is used for the internal nodes and the leaf nodes respectively ($\varepsilon_{node} = \varepsilon_{leaf} = \frac{\varepsilon}{2}$). Rather than selecting the splitting attribute using noisy counts as in [22], Mohammed et al. (2011) propose to use the exponential mechanism (as defined by Theorem 2.6). The main advantage of using the exponential mechanism to select the splitting attribute is that it exponentially favours candidates with a high information gain, hence building more accurate decision trees. In [26], Liu et al. (2018) propose to split ε into s_i shares for the selection of internal nodes, based on the k^{th} harmonic number: $s_i = \frac{1}{k} + \frac{1}{k-1} + \cdots + 1$. Denoting the budget share for leaf nodes by $s_l = 1$, the total number of shares is $s_t = s_i + s_l$. With k being a fixed, pre-defined number of attributes, Liu et al. (2018) assign $\varepsilon_{node} = \frac{\varepsilon}{s_t} * \frac{1}{k^2}$ and $\varepsilon_{leaf} = \frac{\varepsilon}{s_t}$. Under these constraints, the decision tree satisfies ε-differential privacy. We refer the reader to [26] for a formal analysis of the budget allocation. With the above budget allocation strategies and under Theorem 2.7, building an ensemble of T decision trees requires a total privacy budget of εT. This means that if the privacy target is ε, and because of the data overlap amongst the trees, each tree only receives a privacy budget of $\varepsilon_t = \frac{\varepsilon}{T}$. As T grows bigger, trees therefore only receive a very small privacy budget, leading to high noise addition and poor accuracy.

Fletcher et al. (2017) [15] and Li et al. (2020) [25] address this shortcoming by proposing an ensemble of trees where each tree operates on a distinct subset of the data. Under these new constraints, each tree within the ensemble can use the full privacy budget: $\varepsilon_t = \varepsilon$. However, this approach doesn't work well on small datasets. Indeed, since every tree operates on its own subset of data, the total number of instances that each tree receives becomes smaller as the number of trees grows bigger. Consider a dataset of n instances $\mathbf{X} = X_1, \ldots, X_n$ and an ensemble of T trees. Each tree gets roughly $n_{tree} = \lceil \frac{n}{T} \rceil$ instances. If T is too small, then the learning algorithm suffers from the pitfalls outlined in Section 2.1. If T is too high, then n_{tree} is small and the decision trees might not have enough instances to learn the underlying structure of the dataset.

In [41] and [4], the authors mention an adaptive budget allocation method. Rather than fixing an equal privacy budget used at each depth of the tree, the authors propose to compute this quantity during the tree induction algorithm. In the former case, the budget at each depth d is computed as $\varepsilon_{node} = \frac{1}{2^d}\varepsilon_t$. This exponentially decaying budgeting is motivated by the intuition that early splits in the tree are more important than subsequent splits, hence why they get a higher portion of the privacy budget. However, such an allocation method might not be ideal when the tree gets very deep, as lower levels in the tree will receive a really small budget, which in turn creates too much noise and penalizes the quality of the tree. In the latter, the author proposes a new induction algorithm, *ADiffP-φ*, that computes each split's privacy budget dynamically and uses it as a termination criteria when the budget gets too small. Thus, the total number of nodes in the tree is not controlled by the maximum depth (or any other usual termination criteria), but by the total privacy budget available to the tree.

3.1.2 Query sensitivity

Fletcher and Islam (2015) [13] explore reducing the sensitivity of the splitting functions, hopping to reduce the amount of noise added in the tree-building process. Rather than computing the global sensitivity of a function, they propose to compute the local sensitivity of the Gini index in the node being split as $\Delta = 1 - (\frac{n_i}{n_i+1})^2 - (\frac{1}{n_i+1})^2$ where n_i is the support of node i (the support is equal to the number of instances in the node). This sensitivity reduction directly

improves the Laplace and exponential mechanisms, as a tighter sensitivity bound allows for lower added noise. However, in its current form, this sensitivity derivation might be too noisy in some cases and incorrectly providing differential privacy. The same authors later proposed additional work ([14], [15]) based on Nissim et al. (2007) [31]'s smooth sensitivity definition to enhance their model.

In [25], Quinbin et al. (2020) propose a new way to bound the sensitivity of both the leaf nodes and the splitting function. Using new procedures (*Geometric Leaf Clipping* and *Gradient-based Data Filtering)*, they proved that the sensitivity of the leaf nodes could be bound by $\Delta V \leq \min(\frac{g_l^*}{1+\lambda}, 2g_l^*(1-\eta)^{t-1})$ and that the sensitivity of the splitting function could be bound by $\Delta G \leq 3g_l^{*2}$, where g_l^* is the maximum possible 1-norm gradient, λ is a regularisation parameter, η is the learning rate and t is the index of the tree being constructed. These new bounds offer better accuracy in the differentially private settings.

Wang et al. (2020) [41] introduce the first provably accurate privacy preserving, top-down decision tree learning algorithm in the distributed setting. More information about top-down algorithms can be found in [24]. They propose a new splitting function, `PrivateSplit`, that approximates the optimal split chosen by the top-down algorithm, while providing differential privacy and proving boosting-based utility guarantees. In fact, they show that their algorithm only needs $(1/\varepsilon)^{O(\log(1/\varepsilon)/\gamma^2)}$ splits to achieve a training error $\leq \varepsilon$ with probability $\geq 1-\delta$ (provided that the dataset has a certain size). On the adult dataset[2], their algorithm achieves roughly 79% accuracy, which is a little over Quibin et al. (2020) [25]'s accuracy (roughly 74%) over the same dataset and under the same privacy budget constraints $\varepsilon = 1$.

3.1.3 Termination criteria and post-processing

When building a decision tree, one must choose some criteria as to when to stop growing the tree. The simplest criterion is to stop growing the tree once it reaches a certain depth d. Other termination criteria include defining a maximum number of leaf nodes that a tree can have, or defining a minimum number of samples that a node must have before being split (i.e. a minimum node support). However, there is no optimal value for any of these criteria that works for every dataset. Rather, it is left to the user to find appropriate values that work best for their use case. In some cases, these parameters may be estimated based on the dataset. In [12], the authors limit the tree depth to $d = \frac{k}{2}$ where k is the number of features in the dataset. In [22], the tree depth is set to $d = \min(k/2, \log_b(n) - 1)$ where n is the size of the dataset, and b is the branching factor.

Once the tree has reached its termination criteria, one can apply post-processing methods, such as *pruning*, to the tree. Pruning consists in removing untrustworthy leaf nodes from the tree. A leaf node can be considered untrustworthy if its support or purity (homogeneity of the class labels in the node) is too low. In the differentially private settings, pruning cannot be applied as it is in traditional decision trees, as the modifications to the tree induction algorithm induced by differential privacy need to be accounted for. Friedman and Schuster (2010) [17] adapt classical pruning mechanisms by normalising the support of each node in the tree, so that that sum of all counts in all nodes matches the size of the dataset (which is not initially the case due to the noise introduced by differential privacy). In random forests, Fletcher and Islam (2015) [14] observe that in some cases, an additional split worsens the decision tree rather than help it, since the split is chosen at random. They therefore propose to prune leaf nodes where the added noise is greater than the class count signal.

[2] https://archive.ics.uci.edu/ml/datasets/adult

3.1.4 Depth-first and best-leaf first decision tree induction

There are several ways to grow a decision tree during its induction process. While most algorithms grow trees by level, i.e. depth-wise, some implementations (like XGBoost [7]) grow decision trees with a best-leaf first approach. This approach was first described by Haijian S. (2007) [36] in his doctoral thesis. Figure 3.1 and 3.2 show the differences between a depth-first and a best-leaf first tree induction algorithm (the figures are adapted from XGboost documentation[3]).

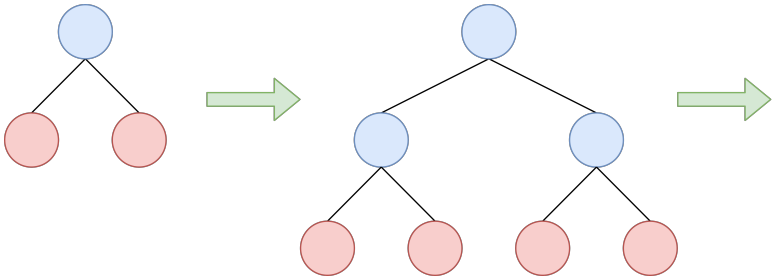


Figure 3.1: A depth-first decision tree induction algorithm.

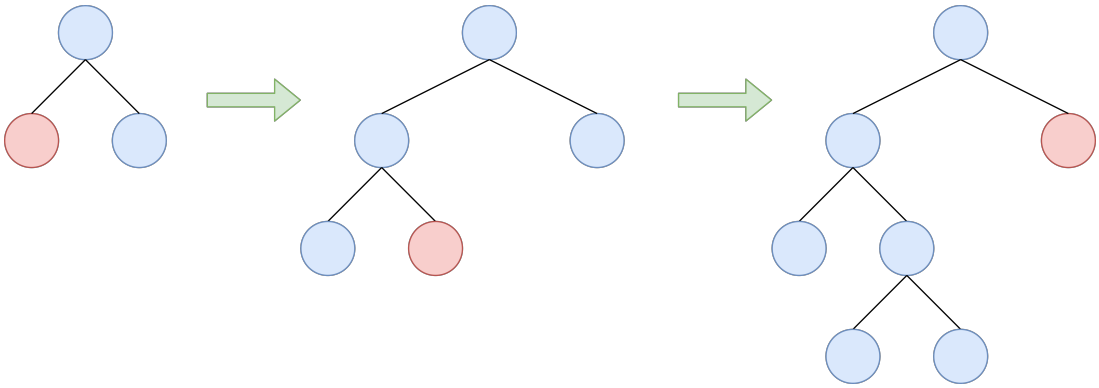


Figure 3.2: A best-leaf first decision tree induction algorithm.

Depending on the dataset, a best-leaf first induction might achieve a lower loss. However, when the number of data is small, a best-leaf first approach may over-fit the dataset.

In the next section, we choose to explore in more details the work of Quibin et al. (2020) [25]. This choice is motivated by the tight sensitivity bounds that the authors were able to derive, along with the good prediction accuracy that their model showed.

3.2 Differentially private gradient boosted decision trees (DP-GBDT)

To make decision trees differentially private with the definitions provided in Section 2.3, the literature ([44], [27], [25]) proposes two changes to Algorithm 1:

- For each node, the attribute and the attribute's value on which the node is split must be chosen through the exponential mechanism. The information gain G is used as the utility function, and the exponential mechanism guarantees that the attributes and attributes' value with higher gain have a higher probability of being chosen.
- For each leaf node, the leaf value must be noised through the Laplace mechanism.

[3] https://github.com/Microsoft/LightGBM/blob/master/docs/Features.rst

Qinbin et al. (2020) [25] suggests that each tree t's privacy budget ε_t gets split into two parts: $\varepsilon_{leaf} = \frac{\varepsilon_t}{2}$ for the leaf nodes and $\varepsilon_{node} = \frac{\varepsilon_t}{2d_{max}}$ for the internal nodes, where d_{max} is the maximum depth for the tree t. Since the nodes in one depth have disjoint inputs, Theorem 2.8 can be applied and the privacy budget in one depth needs to be counted only once. Thus, the total privacy budget consumption is no more than $\varepsilon_{node} * d_{max} + \varepsilon_{leaf} = \varepsilon_t$. Once a tree is trained, the gradients of all instances are updated. This allows the next tree to converge towards a minimum for the objective function defined in Equation 2.1. The authors propose Algorithm 2.

Algorithm 2: Differentially private GBDTs training process [25]

Input: $\mathbf{X} = X_1, \ldots, X_n$: instances, $\mathbf{y} = y_1, \ldots, y_n$: labels
Input: λ: regularisation parameter, d_{max}: maximum depth, η: learning rate
Input: T: total number of trees, l: loss function, ε: privacy budget
Output: An ensemble of trained differentially private decision trees.

$\varepsilon_t = \varepsilon$ ▷ *Each tree is trained on a disjoint subset of the dataset, so we can apply Theorem 2.8*
for $t = 1$ ***to*** T **do**
  Update gradients of all training instances on loss l
  $\varepsilon_{leaf} = \frac{\varepsilon_t}{2}$, $\varepsilon_{node} = \frac{\varepsilon_t}{2d_{max}}$
  for $d = 1$ ***to*** d_{max} **do**
    for *each node in current depth* **do**
      for *each split value* i **do**
        $G_i \leftarrow \frac{(\sum_{i \in I_L} g_i)^2}{|I_L| + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{|I_R| + \lambda}$ ▷ *Equation 2.3*
        $P_i \leftarrow \exp(\frac{\varepsilon_{node} \cdot G_i}{2\Delta G})$ ▷ *Theorem 2.6*
      Split node on split value i, where i is chosen with probability $P_i / \sum_j P_j$
  for *each leaf node* i **do**
    $V_i \leftarrow \eta \left(-\frac{\sum_{i \in I} g_i}{|I| + \lambda} + Lap(0, \Delta V / \varepsilon_{leaf}) \right)$ ▷ *Equation 2.4 and Theorem 2.5*

The authors use the parallel composition theorem (Theorem 2.8) to lower the privacy budget consumption by training each decision tree on a disjoint set of data. However, when the training dataset is small and the number of trees is large, each tree only gets very few samples to learn from. This low number of samples available to the trees results in bad splits, which damages the quality of the trees and their predictions. To address this limitation, we propose in the next chapter a new induction algorithm, *2-nodes*.

Chapter 4

2-nodes Algorithm

Our model was implemented in Python 3, and is available on GitHub[1]. Our base implementation follows [25], with their own implementation (based on XGBoost [7]) available on GitHub as well[2]. We implemented the following features:

- Gradient based data filtering and geometric leaf clipping (from [25], described in Section 3.1.2). This can be enabled or disabled.
- Depth-first, best-leaf first and 2-nodes tree induction, outlined in Section 3.1.4 and in this chapter.
- Decaying privacy budget (from [41], described in Section 3.1.1). This can be enabled or disabled.

4.1 Design

The 2-nodes induction method (Figure 4.1) is a modified version of the classical depth-first tree growth algorithm outlined in Figure 3.1. At each depth d of the tree, for a given node n_{d_i}, $0 \leq d < d_{max}$ and $0 \leq i < 2^d$, we consider the data-points of the sibling node n_{d_j} (j being the index of the node that shares the same parent node as n_{d_i}) while computing the optimal splitting point. The reason we opted for this design is that the exponential mechanism (used to select the splitting point) performs better the more instances there are within a node, as the gain and associated probabilities will be higher (see Equation 2.3). By combining instances in the nodes and their siblings, we can make sure that the chosen splits will fit the data better.

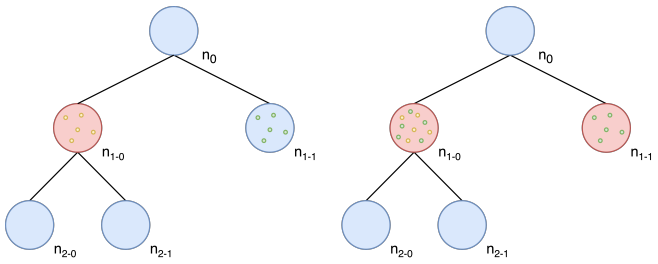


Figure 4.1: Data-points (red nodes) being considered while splitting a node for the depth-first (left) vs. 2-nodes (right) algorithms. In the depth-first case, node n_{1_0}'s best split is computed over the instances of the node itself. In the 2-nodes case, node n_{1_0}'s best split is computed on the instances of the node itself, and the instances of its sibling node n_{1_1}.

[1] https://github.com/giovannt0/dpgbdt
[2] https://github.com/QinbinLi/DPBoost

4.2 Properties

In a differentially private scenario, the data-points in each node (except the root node) at every level will be queried twice (once when splitting the node, and once when splitting its sibling node). We can therefore adapt Algorithm 2 into Algorithm 3.

Algorithm 3: 2-nodes DPGBDT training process

Input: $\mathbf{X} = X_1, \ldots, X_n$: instances, $\mathbf{y} = y_1, \ldots, y_n$: labels
Input: λ: regularisation parameter, d_{max}: maximum depth, η: learning rate
Input: T: total number of trees, l: loss function, ε: privacy budget
Output: An ensemble of trained differentially private decision trees.

$\varepsilon_t = \varepsilon$ ▷ *Each tree is trained on a disjoint subset of the dataset, so we can apply Theorem 2.8*
for $t = 1$ ***to*** T **do**
  Update gradients of all training instances on loss l
  $\varepsilon_{leaf} = \frac{\varepsilon_t}{2}$, $\varepsilon_{node} = \frac{\varepsilon_t}{4d_{max}}$ ▷ *The budget for internal nodes is half of that in Algorithm 2*
  for $d = 1$ ***to*** d_{max} **do**
    for *each node in current depth* **do**
      $g = concat(g_{node}, g_{node\_sibling})$ ▷ *Concat the gradients of the node and its sibling*
      for *each split value* i **do**
        $G_i \leftarrow \frac{(\sum_{i \in I_L} g_i)^2}{|I_L| + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{|I_R| + \lambda}$ ▷ *Equation 2.3*
        $P_i \leftarrow \exp(\frac{\varepsilon_{node} \cdot G_i}{2\Delta G})$ ▷ *Theorem 2.6*
      Split node on split value i, where i is chosen with probability $P_i / \sum_j P_j$
  for *each leaf node* i **do**
    $V_i \leftarrow \eta \left(-\frac{\sum_{i \in I} g_i}{|I| + \lambda} + Lap(0, \Delta V / \varepsilon_{leaf}) \right)$ ▷ *Equation 2.4 and Theorem 2.5*

Theorem 4.1 *The output of Algorithm 3 satisfies ε-differential privacy.*

Proof Let ε be the total privacy budget for a gradient boosted decision tree model of T trees. Since the trees receive a disjoint subset of the dataset, each tree receives a budget of $\varepsilon_t = \varepsilon$. Let $\varepsilon_{leaf} = \frac{\varepsilon_t}{2}$ and $\varepsilon_{node} = \frac{\varepsilon_t}{4d_{max}}$ be the budgets for tree t's leaf nodes and its internal nodes. At each depth of the tree t, the inputs are disjoints and are queried exactly twice.

Per Theorem 2.7 and 2.8, the privacy budget consumption for a single tree t of depth d_{max} does not exceed $\varepsilon_{leaf} + 2 * d_{max} * \varepsilon_{node} = \frac{\varepsilon_t}{2} + 2 * d_{max} * \frac{\varepsilon_t}{4 * d_{max}} = \varepsilon_t$. This holds for all T trees, as per Theorem 2.8. The budget consumption therefore never exceeds ε. □

Legitimately, the above algorithm raises questions regarding choosing 2 nodes to compute the gains and not a greater number. There are two main reasons for not choosing a number that is greater than 2:

1. Since we query the instances n times, n being the number of nodes taken into account during node splitting, the privacy budget needs to be divided by n as per Theorem 2.7. For a small ε and a large n, the resulting privacy budget that is left for the nodes would be so small that there would be too much added noise in the resulting tree. This would lead to bad splits, itself leading to bad predictions.

2. In 2-nodes, we consider the direct sibling node. The node being split and its sibling share the same parent node. This means that, up to this point in the tree, instances in both nodes are very similar, thus they can be worked on together and lead to good results. If we were to take other nodes into account, in other sub-trees that are at the same depth

as the node being split, then the instances in these nodes might differ significantly. This is the case when an earlier split in the tree happens on a very distinctive attribute of the dataset.

A detailed evaluation of the performances of the 2-nodes induction algorithm for both real life and synthetic datasets is given in Section 6.

Chapter 5

Synthetic Data

There are many scenarios where privacy preserving machine learning is desirable. For instance, consider a dataset which contains patient data, such as their health history. A leak of this dataset would be very bad for the patient's privacy, as the disclosed information could be used to discriminate against them (e.g. higher health insurance premiums). This risk can make patients more hesitant to share their data. This in turn causes the problem of having small datasets.

In the cyber risk scenario, historical data is lacking. This is because many insurance companies have just started to offer this line of product. While we were able to get a few anonymised cyber insurance questionnaires from our industry partner[1], it is not enough to properly evaluate our model. This is the motivation of this chapter, in which we generate our own synthetic datasets. For our purpose, we introduce two distinct targets: *loss*, which is a probability $p \in [0,1]$ that a company suffers a loss, and *cost*, which is a positive number $cost \in \mathbb{R}^+$ that quantifies the losses of the company following a security incident. While the process tries to mimic real data as much as possible, there is no ground truth for the type of data that we are generating. As such, we acknowledge that we cannot verify the quality or truthfulness of the generated datasets. Figure 5.1 describes the overall process of data generation.

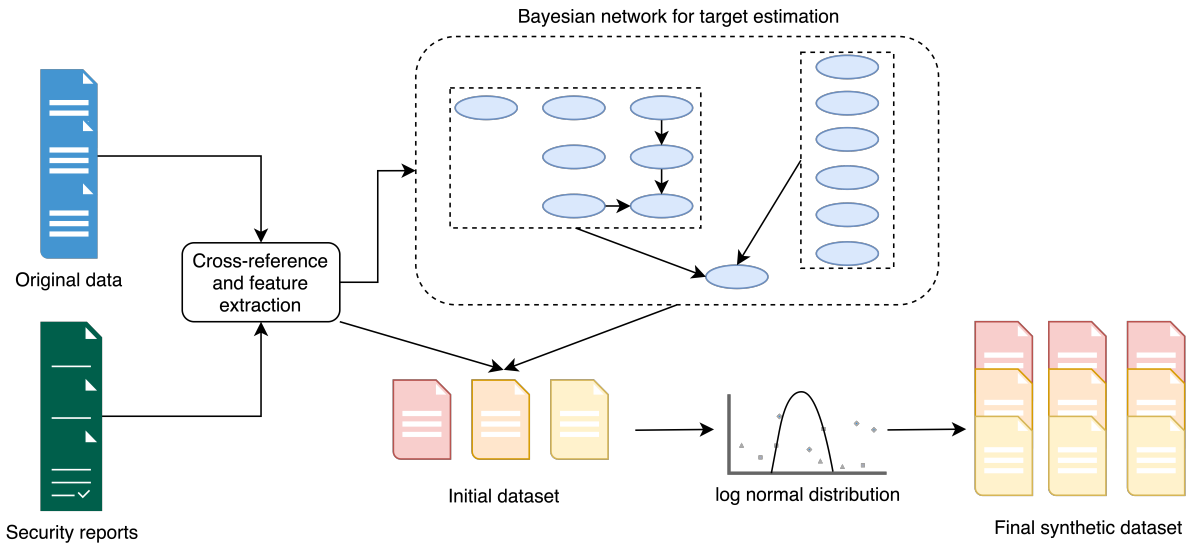


Figure 5.1: Synthetic data generation process.

[1] https://zisc.ethz.ch/research/projects/privacy-preserving-machine-learning-for-cyber-insurance/

In a first part, we cross-reference the few questionnaires we were given with various security reports, and extract our dataset's features. This is presented in Section 5.1. In a second part, we build a Bayesian network based on the extracted features to estimate our dataset's targets. This is described in Section 5.2. Finally, we use a log normal distribution to generate samples that share the same features but that have different target values. This is covered in Section 5.3.

5.1 Features extraction

There are two components to any dataset: a set of features and a set of targets. The former describes attributes of the dataset's instances, while the latter captures the goal of the prediction task. The set of features and the set of targets should be closely related, otherwise the learning task becomes quite meaningless. Consider for instance the adult[2] dataset, where the prediction task is to determine whether or not a person makes over $50k$\$ a year. The set of features is composed of various attributes about the persons: their age, education, workclass, etc. These features are directly relevant to the prediction task, since they will influence the target value: a person that graduated high school versus a person that graduated from a Doctorate degree are likely to have a measurable income gap.

To identify our features, we need to understand how companies handle security, i.e. what are their security practices, and what are the implications on the *loss* and *cost* targets. These security practices can be technical (e.g. firewall configuration) or operational (e.g. does the company have a security and privacy policy that their employees must adhere to). Unfortunately, companies are likely to be concerned that honest answers indicating poor IT security practices could be used to discriminate against them. Hence, they are not very keen on releasing such information.

To shed some light on this, we went through annually released security reports from various vendors: Verizon [40], NetDiligence [29], IBM [21], Cisco [9], Checkpoint [5], and AIG [1]. These reports highlight current security practices that company follow (or, fail to follow) and provide insights about the threat landscape they face, in an anonymised fashion. Of particular relevance to our business problem, the NetDiligence [29] and AIG [1] reports provide data from the insurer's perspective, highlighting common causes and consequences about cyber attacks that their clients suffered.

Using these reports, we can derive useful probabilities. NetDiligence [29] shows that in the $2014-2018$ period, social engineering (act of tricking someone into divulging information or taking action) was responsible for 28% of the claims they received from companies. Since this number is post-incident, we can transform it into a conditional probability between the event *social engineering* and the event *the company suffers a loss*: $Pr(social\_engineering|loss) = 0.28$. To benefit our synthetic data generation problem, we can tweak this a little and create a new feature, f_1 = *company_trains_employees_against_social_engineering*. We can now artificially derive: $Pr(f_1|loss) = 1 - Pr(social\_engineering|loss) = 1 - 0.28 = 0.72$. It could be argued that for some companies, training was delivered but improperly, thus raising questions about the probability that was derived. For this thesis and for the sake of simplicity, we will assume an ideal scenario.

Following a similar approach, we derive multiple such features and probabilities across all the reports, which we summarise in Table 5.1. If multiple vendors report about the same incident (e.g. social engineering), we take the mean value that was reported across all reports. As these reports do not provide pre-incident figures (e.g., $Pr(f_1)$), we manually assign a probability to such events, where relevant. We additionally create the following features for which

[2] https://archive.ics.uci.edu/ml/datasets/adult

the reports did not provide any information: *company_uses_3rd_party*, *company_has_ids_ips*, *company_has_recovery_plan*, *company_has_firewalls*. For the company sector, we create a single feature *sector* which can be one of $[hospitality, manufacturing, energy, \ldots]$. We let the reader refer to Table A.1 for more information about the features and their meaning.

While the companies' security practices will likely influence the *loss* target, not all features influence the *cost* target. Typically, f_1 = *social_engineering* is not expected to contribute to the *cost* target, as this target is post-incident: employees being trained against social engineering is not expected to influence how much the company will lose once an incident has happened. To make up the *cost* target, we need to position ourselves on the attacker's side. Once an initial foothold inside the company has been established, the kind of data they may be able to find is likely to be relevant, as it will allow them to progress their attack further.

The Verizon report [40] gives us a breakdown of the type of data that each industry is likely to possess: PII (personal identifiable information), PCI (payment card industry), PHI (protected health information), credentials (passwords), or 'other'. This is reported in Table 5.2. This allows us to derive further features: *company_has_pii*, *company_has_pci*, *company_has_phi*, *etc*.

Feature (F)	$Pr(F)$	$Pr(F\|loss)$	$Pr(F\|SME)$	$Pr(F\|large)$
$company\_is\_in\_hospitality$			0.03	0.04
$company\_is\_a\_public\_entity$			0.03	
$company\_is\_in\_other$			0.11	0.09
$company\_is\_a\_nonprofit$			0.05	
$company\_is\_in\_education$			0.05	0.08
$company\_is\_in\_technology$			0.06	0.03
$company\_is\_in\_manufacturing$			0.08	0.03
$company\_is\_in\_financial\_services$			0.09	0.15
$company\_is\_in\_energy$				0.04
$company\_is\_in\_retail$			0.09	0.24
$company\_is\_in\_healthcare$			0.19	0.26
$company\_is\_in\_professional\_services$			0.22	0.04
$company\_trains\_employees\_against\_social\_engineering$	0.70	0.8025		
$company\_has\_patching\_process$	0.60	0.828		
$company\_has\_antivirus$	0.95	0.81		
$company\_uses\_3rd\_party$	0.87			
$company\_has\_ids\_ips$	0.94			
$company\_has\_recovery\_plan$	0.33			
$company\_has\_firewalls$	0.83			

Table 5.1: Potential features for the synthetic dataset and their respective probabilities. We separate small and medium sized businesses (SME) from large businesses.

5.2 Targets estimation

Section 5.1 introduced a way to identify the set of features for our synthetic dataset, and their corresponding probabilities for feature selection. Feature selection consists in deciding if a feature is *True* or *False* for a given company. For instance, the social engineering feature f_1 has a probability of $Pr(f_1) = 0.70$ according to Table 5.1. During the data generation process, each

Sector (S)	$Pr(PII\|S)$	$Pr(PCI\|S)$	$Pr(PHI\|S)$	$Pr(credentials\|S)$	$Pr(other\|S)$
Hospitality	0.44			0.34	0.23
Public Entity	0.51			0.33	0.34
Other	0.81			0.36	0.42
Education	0.75			0.30	0.23
Technology	0.69			0.41	0.34
Manufacturing	0.49	0.20		0.55	0.25
Financial Services	0.77	0.32		0.35	0.35
Energy	0.41	0.68		0.41	0.35
Retail	0.49	0.47		0.27	0.25
Healthcare	0.77		0.67	0.18	0.18
Professional Services	0.75			0.45	0.32

Table 5.2: Probabilities for companies within a specific sector to store a certain type of data

instance will see this feature set to *True* with probability 0.70, and to *False* with probability 0.30. This process of feature selection is repeated across all features. Once this is done, we must use the results of the feature selection process to compute fictive values for our targets *loss* and *cost*. This section presents two approaches, both using Bayesian networks, that vary in complexity.

Definition 5.1 (Graphical model [39]) *A graphical model is a tool that is used to visually illustrate and work with conditional independencies among variables in a given problem.*

A graph is composed of a set of nodes (which represent the variables) and a set of edges. Each edge connects two nodes, and an edge can have an optional direction associated to it. Two variables are conditionally independent if they have no direct impact on each other's value. Figure 5.2 shows an example of a graphical model.

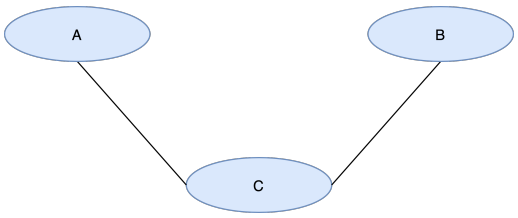


Figure 5.2: A graphical mode, in which A and B are conditionally independent given C.

Definition 5.2 (Bayesian network [39]) *Bayesian networks are a particular instance of graphical models. They are directed acyclic graphs (DAG): all edges in the graph are directed (i.e. they point in a particular direction) and there are no cycles (i.e. there is no way to start from a node, travel along a set of directed edges and arrive back at the starting node).*

Figure 5.3 illustrates a Bayesian network. Given nodes $\mathbf{X} = X_1, \ldots, X_n$, the joint probability function for any Bayesian network is $Pr(\mathbf{X}) = \prod_{i=1}^{n} Pr(X_i|parents(X_i))$. This means that the joint probability of all the variables is the product of the probabilities of each variable given its parents' value. In Figure 5.3, we have $Pr(A, B, C) = Pr(A)Pr(B)Pr(C|A, B)$.

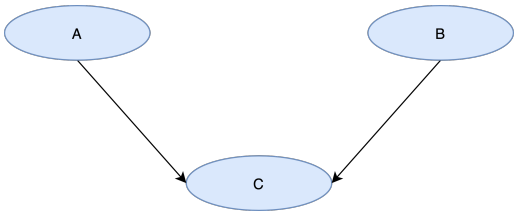


Figure 5.3: A simple Bayesian network.

5.2.1 Approach 1

Loss target

For the *loss* target, we propose the Bayesian network outlined in Figure 5.4. We make the *loss* target depend on a subset of the features: *social engineering training*, *antivirus*, and *outdated software patching process*. Since the reports do not mention any correlations between these features, we assume that they are conditionally independent with respect to *loss*.

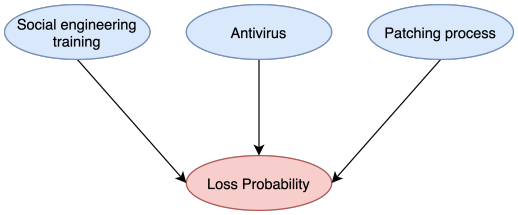


Figure 5.4: Bayesian network for the *loss* target.

We can then derive:

$$\begin{aligned} Pr(loss|\ social, antivirus, patching) &= \frac{Pr(loss, social, antivirus, patching)}{Pr(social)Pr(antivirus)Pr(patching)} \\ &= \frac{Pr(social, antivirus, patching|loss)Pr(loss)}{Pr(social)Pr(antivirus)Pr(patching)} \\ &= \frac{Pr(social|loss)Pr(antivirus|loss)Pr(patching|loss)}{Pr(social)Pr(antivirus)Pr(patching)} \end{aligned} \tag{5.1}$$

Fixing $Pr(loss) = 0.10$ and plugging in the data provided in Table 5.1, we can compute the *loss* target.

Cost target

To derive a number for the target *cost*, we exclusively rely on data provided in the NetDiligence [29] report:

- The `min`, `max` and `mean` cost of a breach in a particular industry: $min(cost_{breach})$, $max(cost_{breach})$, $mean(cost_{breach})$
- The `mean` post-incident (crisis) cost, i.e. the breach remediation cost $cost_{remediation}$
- The `mean` cost per type of data that was potentially exposed, $cost_{data}$. This cost directly depends on the features of a given instance: if the instance is in healthcare, then as per Table 5.2, we will add together the costs for leaked PII, PHI, credentials and others if these attributes are *True* for that instance.

In particular, we derive the cost as per the following formula: $cost = triangular(min(cost_{breach}), max(cost_{breach}), mean(cost_{breach})) + cost_{remediation} + \sum_{data} cost_{data}$ where $triangular()$ is the triangular distribution. Fig. 5.5 shows the distribution of values for the targets over 1 million generated samples. The method used to generate the samples is detailed in Section 5.3.

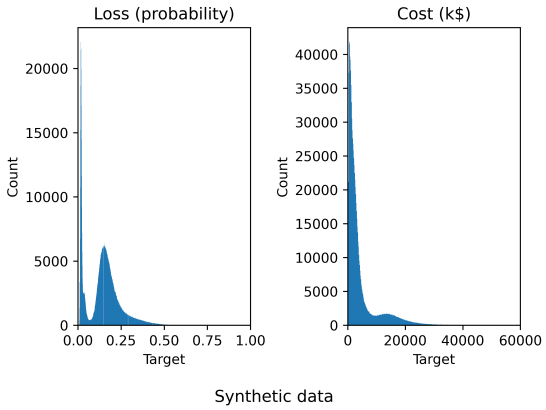


Figure 5.5: Target distribution for the synthetic dataset.

Unfortunately, our testing demonstrated that the problem was rendered hard through too much randomness / noise addition, rather than by the intricacies of the features and the targets themselves. This is why we now propose a second approach to the problem.

5.2.2 Approach 2

Loss target

For the *loss* target, we propose a more complex Bayesian network, as depicted in Figure 5.6. Table A.2 summarises the probabilities that we used in this model. Following the logic of Equation 5.1, we can derive:

$$\begin{aligned}
Pr(A) &= Pr(company_size|company_sector)Pr(company_size) \\
Pr(A|loss) &= Pr(company_size, company_sector|\ loss) \\
Pr(B) &= Pr(BYOD)Pr(pentest)Pr(RBAC)Pr(red_teaming)Pr(2FA)Pr(pwd_policy|2FA) \\
&\quad Pr(social|pwd_policy, red_teaming) \\
Pr(B|loss) &= Pr(BYOD|loss)Pr(pentest|loss) \\
&\quad Pr(RBAC|loss)Pr(red_teaming, 2FA, pwd_policy, soc|loss) \\
Pr(C) &= Pr(patching)Pr(antivirus)Pr(fw_ingress)Pr(fw_egress)Pr(3rd_party)Pr(CISO) \\
Pr(C|loss) &= Pr(patching|loss)Pr(antivirus|loss)Pr(fw_ingress|loss)Pr(fw_egress|loss) \\
&\quad Pr(3rd_party|loss)Pr(CISO|loss) \\
Pr(D) &= Pr(IPS|IDS)Pr(IDS)Pr(threat|IDS)Pr(IOC|threat) \\
Pr(D|loss) &= Pr(IPS, IDS, threat, IOC|loss)
\end{aligned} \tag{5.2}$$

Which we can plug into:

$$Pr(loss|A, B, C, D) = \frac{Pr(A|loss)Pr(B|loss)Pr(C|loss)Pr(D|loss)Pr(loss)}{Pr(A)Pr(B)Pr(C)Pr(D)} \tag{5.3}$$

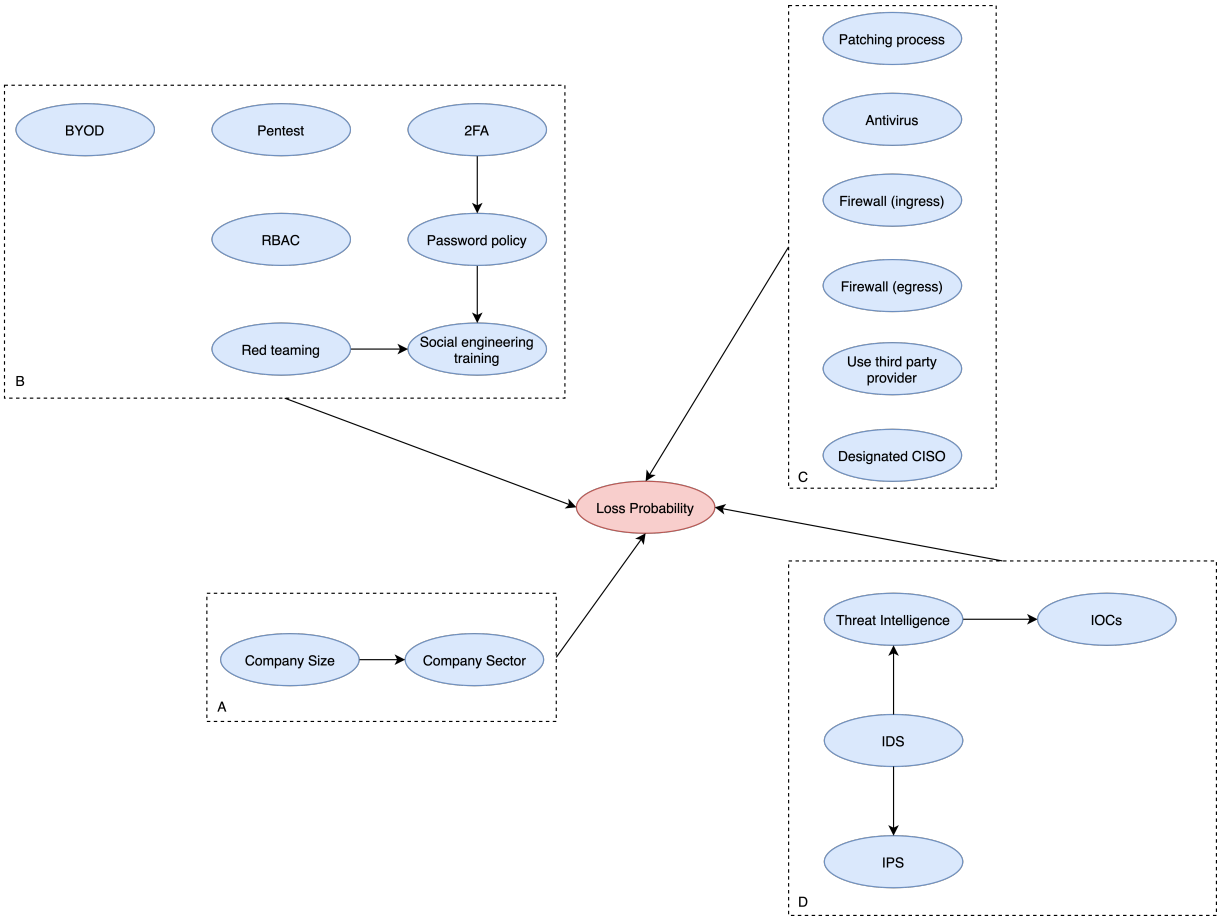


Figure 5.6: Revisited Bayesian network for the *loss* target.

Cost target

For the *cost* target, we propose additional features that influence the outcome, as shown in Figure 5.7. The cost is now computed as:

$$cost = \sum_{data} cost_{data} + \lambda_1 mean(cost_{breach}) + \lambda_2 cost_{remediation} \tag{5.4}$$

where λ_1 and $\lambda_2 \in [0, 1]$ represent the fraction of the cost that should be applied to the instance. They are computed at runtime depending on the instance's features. For example, if the instance has the feature *company_has_incident_response_team* set to *True*, then we decrease λ_2 by a certain amount, as we expect the remediation cost to be lower in the presence of an IR team.

We generate 4 synthetic datasets, each of 1 million samples, using this new approach. The distributions of the targets are depicted in Figure A.1.

5.3 Data generation

We start the data generation process by first creating what we called an *instance stereotype*. An instance stereotype is an instance with a defined set of features (e.g. $company\_has\_antivirus : True$, $company\_has\_IDS : True$, $company\_has\_IPS : False$, ...). In practice, we expect that instances may share the same features while having slightly different target values. This is because there are many other factors that influence the *loss* and *cost* targets in real life, which

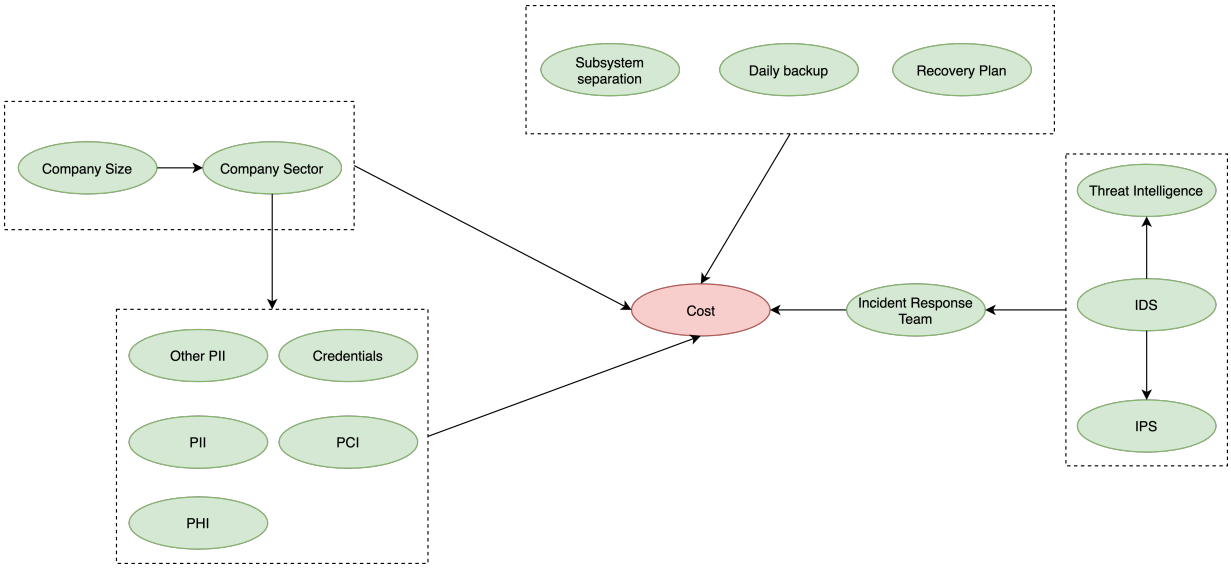


Figure 5.7: Features that influence the *cost* target.

are not captured here. For instance, two companies may both have an incident response team but their process may differ, and incident containment could take the first company two hours and the second one two days, which may in turn impact the *cost* target. To model these differences, we rely on a log normal distribution.

Definition 5.3 (Log normal distribution) *Let Z be a standard normal variable, and let μ and $\sigma > 0$ be two real numbers. Then, the distribution of the random variable $X = e^{\mu+\sigma Z}$ is called the log-normal distribution with parameters μ and σ.*

***Note**: μ and σ are the parameters of the variable's natural logarithm, not of X itself. In order to produce a distribution with desired mean μ_X and standard deviation σ_X, one uses $\mu = \ln\left(\frac{\mu_X^2}{\sqrt{(\mu_X^2+\sigma_X^2)}}\right)$ and $\sigma = \sqrt{\ln\left(1+\frac{\sigma_X^2}{\mu_X^2}\right)}$.*

We start with some instance stereotype and derive additional instances that share the same features but slightly different target values: the target values are now drawn from the log normal distribution. For example, if we have an instance stereotype with *loss* probability 0.30, we model a log normal distribution with mean $\mu_X = 0.30$ and standard deviation $\sigma_X = 0.30/100 * 15$ (i.e. 15% deviation). Instances that stem from this instance stereotype will have their *loss* target drawn at random from the associated log normal distribution. Each stereotype will contribute to generating 1% of the total number of samples. Algorithm 4 details the procedure to generate the synthetic datasets.

We evaluate the datasets through 3-fold cross-validation using our non-DP (vanilla) gradient boosting algorithm. We set the learning rate to 0.5, and the maximum depth to 15. The datasets are evaluated for [300, 5000, 15000, 25000, 50000, 75000, 100000] samples. The number of trees is fixed to 20. Figure A.2 reports the mean absolute percentage error (MAPE) obtained with the baseline model over all synthetic datasets, for both the *loss* and *cost* targets. Table 5.3 reports the mean absolute percentage error rate for each dataset and target, where $n\_samples = 300$ is ignored so that the mean isn't skewed upward, as the learning task is much more difficult with this low amount of samples.

Algorithm 4: Synthetic data generation

Input: *n_samples*: number of samples to generate
Output: *n_samples* synthetic data points

$samples \leftarrow []$
while *len(samples)* $<$ *n_samples* **do**
  $stereotype \leftarrow$ **NewStereotype()**
  $n\_stereotype \leftarrow \lceil n\_samples/100 \rceil$
  $samples\_stereotype \leftarrow$ **GenerateFromStereotype(***stereotype*, *n_stereotype***)**
  $samples \leftarrow samples$.**extend(***samples_stereotype***)**
return $samples[: n\_samples]$

function NewStereotype()
  $profile \leftarrow$ **Init()** ▷ *Initialise attributes based on Table A.2 and 5.2*
  $profile.\texttt{loss} \leftarrow Pr(loss|A, B, C, D)$ ▷ *Equation 5.3*
  $profile.\texttt{cost} \leftarrow \sum_{data} cost_{data} + \lambda_1 mean(cost_{breach}) + \lambda_2 cost_{remediation}$ ▷ *Equation 5.4*
  return *profile*

function GenerateFromStereotype(*stereotype*: `Stereotype`, *n_stereotype*: `int`)
  $samples \leftarrow []$
  $\mu_{loss},\ \sigma_{loss} = \ln\left(\frac{stereotype.\texttt{loss}^2}{\sqrt{(stereotype.\texttt{loss}^2 + (stereotype.\texttt{loss} * 0.15)^2)}}\right),\ \sqrt{\ln(1 + \frac{(stereotype.\texttt{loss} * 0.15)^2}{stereotype.\texttt{loss}^2})}$
  $\mu_{cost},\ \sigma_{cost} = \ln\left(\frac{stereotype.\texttt{cost}^2}{\sqrt{(stereotype.\texttt{cost}^2 + (stereotype.\texttt{cost} * 0.15)^2)}}\right),\ \sqrt{\ln(1 + \frac{(stereotype.\texttt{cost} * 0.15)^2}{stereotype.\texttt{cost}^2})}$
  for $\_ = 1$ ***to*** *n_stereotype* **do**
    $stereotype.\texttt{loss} \leftarrow$ **LogNormal(**μ_{loss}, σ_{loss}**)**
    $stereotype.\texttt{cost} \leftarrow$ **LogNormal(**μ_{cost}, σ_{cost}**)**
    $samples$.**append(***stereotype***)**
  return *samples*

Dataset	Target: *loss*	Target: *cost*
Synthetic A	3.44 ± 0.11	2.08 ± 0.21
Synthetic B	2.28 ± 0.13	1.85 ± 0.04
Synthetic C	2.22 ± 0.12	1.82 ± 0.02
Synthetic D	2.06 ± 0.08	1.85 ± 0.04

Table 5.3: Mean Absolute Percentage Error (%) for the synthetic datasets.

Chapter 6

Performance Evaluation

In this chapter, we evaluate the performances of non-private GBDT and differentially private GBDT. For the latter, we compare our 2-nodes algorithm versus the classic depth-first tree induction algorithm. We use 3 real datasets not related to cyber insurance, and 4 synthetic cyber insurance datasets (since no real publicly released datasets are available). For the real datasets, we follow the literature and report the root mean square error (RMSE) for regression tasks, and the test error (in %) for classification tasks. For the synthetic datasets, we report the mean absolute percentage error (MAPE, in %).

6.1 Datasets

We evaluate the different models on the following datasets:

- Abalone[1]: each instance describes attributes of an abalone, such as their sex, length, or weight. The prediction task is a regression task, consisting in predicting the age of the abalone. There are 8 features and 4177 instances.
- YearPredictionMSD[2]: each instance describes audio features. The prediction task in a regression task, consisting in predicting the year of release of the song. There are 90 features and 515345 instances.
- Adult[3]: each instance gives information about the background and education of a person. The prediction task is a classification task, consisting in predicting whether a person makes more than $50k$\$ a year or not. There are 14 features and 48842 instances.
- Synthetic_{A, B, C, D}: synthetic datasets generated according to Algorithm 4. Each dataset contains 1 million samples, and there are 28 features per instance. The prediction task is a regression task (refer to Chapter 5 for more details).

For each dataset, we take a sample of $n = 5000$ instances. For the real datasets, we fix the tree depth to $d_{max} = 6$, the learning rate to $\eta = 0.1$ and the number of trees to $n_{trees} = 50$. For the synthetic datasets, we fix $d_{max} = 15$, $\eta = 0.5$ and $n_{trees} = 20$. The models are evaluated through a 5-cross validation process. We report results for the vanilla GBDT (non-private, as in Algorithm 1) and the differentially private GBDT algorithms (depth-first and 2-nodes variants, as in Algorithm 2 and 3 respectively).

[1] https://archive.ics.uci.edu/ml/datasets/abalone
[2] https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD
[3] https://archive.ics.uci.edu/ml/datasets/adult

6.2 Results

We use values of ε in the $[0.1,\ 1.[$ range. Intuitively, the higher the privacy budget ε is, the less noise will be added to the model. This results in better accuracy, but in lower privacy guarantees. We choose not to evaluate the model on higher privacy budgets, as the existing literature (e.g. [25]) reports great accuracy in such settings already.

Table 6.1 shows the mean training time (in seconds) per tree and per dataset. The depth-first variant implements the algorithm outlined in [25], while the 2-nodes variant tries to enhance it. For the Abalone and YearPredictionMSD datasets, we report the root mean square error (RMSE) for our model. For the Adult dataset, we report the test error (in %).

	(Non-DP) Vanilla	(DP) Depth-first	(DP) 2-nodes
Abalone	2.44	0.26	0.33
YearPredictionMSD	85.3	11.92	12.06
Adult	0.87	0.17	0.16
Synthetic A	0.37	0.40	0.42
Synthetic B	0.30	0.41	0.41
Synthetic C	0.68	0.41	0.41
Synthetic D	0.27	0.40	0.40

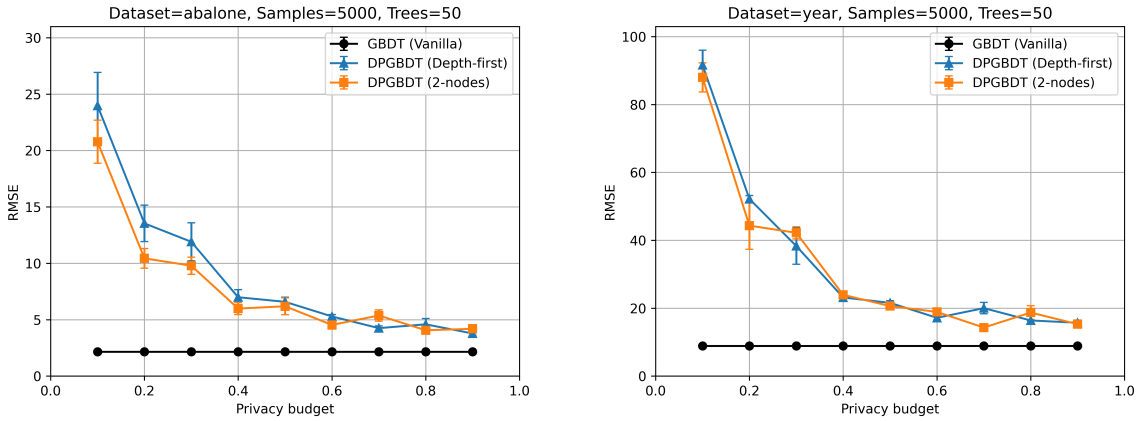
Table 6.1: Mean training time (in seconds) per decision tree, for all datasets.

As shown in Figure 6.1, our 2-nodes induction method performs slightly better than the regular depth-first induction method. This is especially visible in the Abalone dataset when ε is low and when we decrease the number of samples to $n = 300$ and the number of trees to $n_{trees} = 5$, where for $\varepsilon = 0.1$ our *2-nodes* model performs 31.94% better than the regular depth-first model. (reported in Figure 6.2).

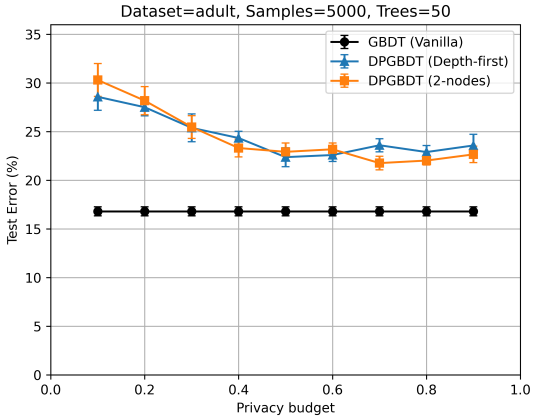
Table 6.2 summarises the results for the vanilla model versus the differentially private model (where the privacy budget is fixed to $\varepsilon = 0.5$) on these real-life datasets.

	(Non-DP) Vanilla	(DP) Depth-first	(DP) 2-nodes
Abalone (RMSE)	2.15 ± 0.04	6.58 ± 0.40	$\mathbf{6.18 \pm 0.75}$
YearPredictionMSD (RMSE)	8.87 ± 0.16	21.57 ± 0.64	$\mathbf{20.62 \pm 0.13}$
Adult (Test error)	16.81 ± 0.45	$\mathbf{22.37 \pm 0.97}$	22.92 ± 0.91

Table 6.2: Prediction error for the real datasets.



(a) Regression tasks: Abalone and YearPredictionMSD



(b) Classification task: Adult

Figure 6.1: Prediction error for the vanilla model and various values of ε for the differentially private model.

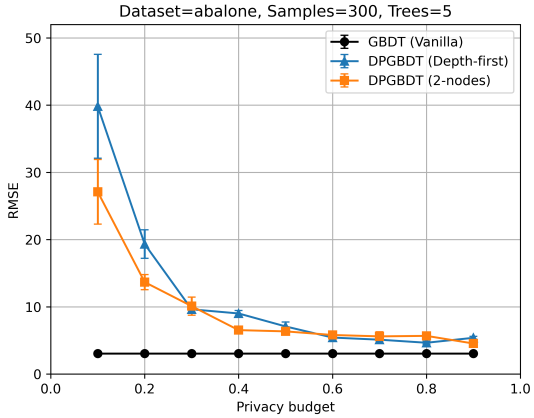


Figure 6.2: Prediction error for the Abalone dataset, with 300 samples.

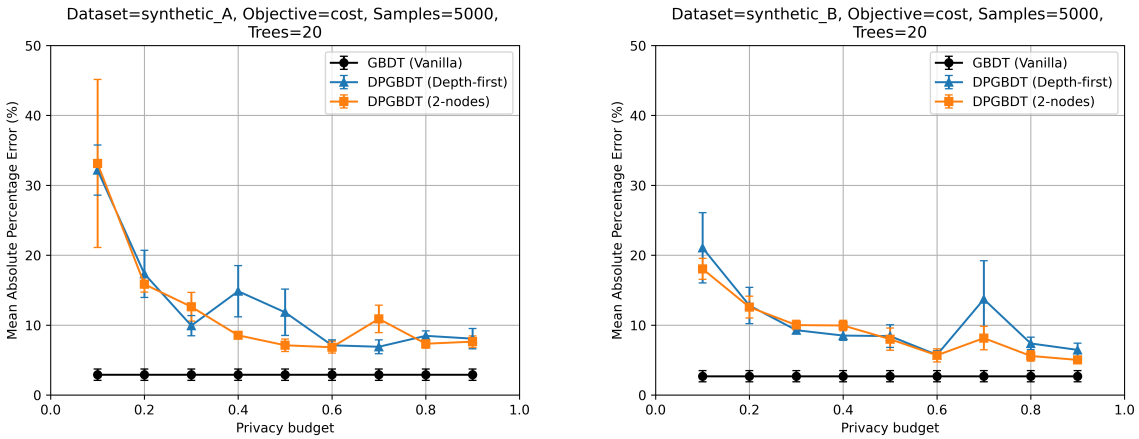
For the synthetic datasets, we report the MAPE, defined as $MAPE = \frac{1}{n}\sum_i^n |\frac{R_i - P_i}{R_i}|$ where R_i is the real value and P_i the predicted one. If the prediction task was to predict the price of an object, with its real price being 100\$ and the predicted price being 110\$, then the MAPE score would be $MAPE = |\frac{100-110}{100}| = 0.1$ i.e. 10% error, since we would be 10\$ off.

Results for the *cost* target are reported in Figure 6.3. For the *loss* target, the reader can refer to Figure A.3. Table 6.3 summarises the results for the vanilla model versus the differentially private model (where the privacy budget is fixed to $\varepsilon = 0.5$), for the *cost* target. For the *loss* target, the reader can refer to Table A.3.

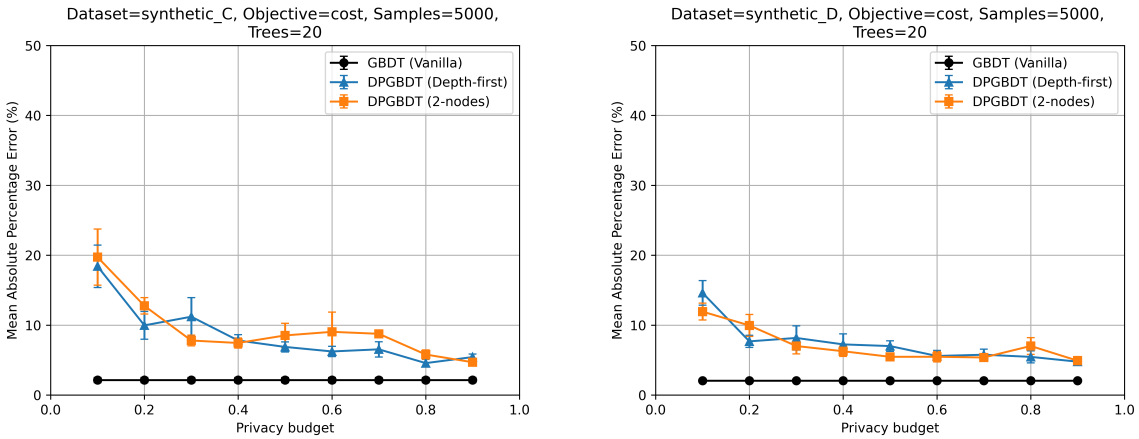
	(Non-DP) Vanilla	(DP) Depth-first	(DP) 2-nodes
Synthetic A (MAPE)	2.90 ± 0.80	11.83 ± 3.31	$\mathbf{7.12 \pm 0.89}$
Synthetic B (MAPE)	2.69 ± 0.79	8.43 ± 1.62	$\mathbf{7.99 \pm 1.58}$
Synthetic C (MAPE)	2.13 ± 0.36	$\mathbf{6.88 \pm 0.71}$	8.52 ± 1.74
Synthetic D (MAPE)	2.05 ± 0.16	7.00 ± 0.76	$\mathbf{5.47 \pm 0.42}$

Table 6.3: Prediction error for the *cost* target on the synthetic datasets.

For the synthetic dataset A, 2-nodes is able to decrease the error by almost 40%. For datasets B and D, the error decreases by 5.22% and 27.97% respectively. On dataset C, 2-nodes performs worst with an error increase of 23.84%.



(a) Synthetic datasets A and B



(b) Synthetic datasets C and D

Figure 6.3: Mean Absolute Percentage Error for the *cost* target on the synthetic datasets.

Chapter 7

Security Analysis

7.1 Attack landscape

In this section, we give an overview of the various attacks that we consider for our model.

7.1.1 Enclave attacks

Most SGX attacks are side-channel [30] attacks, i.e. attacks that are based on information gained from the implementation of the system under attack, rather than weaknesses in the implementation itself. Popular side-channel attacks against SGX are cache-based timing attacks, such as *Flush+Reload* [43] or *Prime+Probe* [32]. Both attacks exploit cache behaviour to leak information on victim access to shared memory. In *Flush+Reload*, the attacker flushes a memory line and then measures the time that it takes for the line to be reloaded. If the line was reloaded fast, then the attacker infers that the victim accessed the data located at that line. In *Prime+Probe*, the attacker first primes the cache (i.e. loads it with dummy data) and waits for the victim to access one of the cache lines. Afterwards, the attacker probes the cache and measures its response time. If the access is fast, then the victim did not access this cache line. If it is slow, it did access it. Figure 7.1 shows how processors fetch data from the cache / memory, and how it relates to access time.

To defend against such attacks, the enclave's authors must make sure that their design is side-channel resilient. This can be achieved by making sure that the code is designed in a cache leakage-free manner, making the execution flow and memory access patterns independent of the data accessed. For other kinds of attacks, [30] suggests that authors can act on multiple fronts: microcode patches, system/application design and compiler/SDK. While this is an active and interesting research area, it is mostly independent of the focus of this thesis, and thus we will not explore these attacks further.

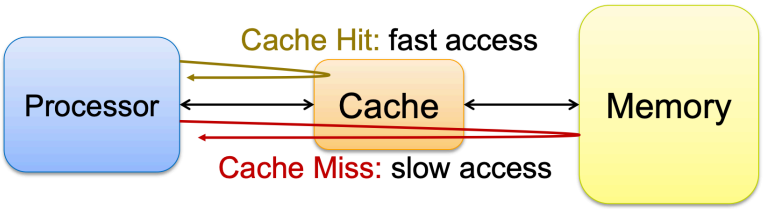


Figure 7.1: Illustration of a processor fetching data from cache or memory [19].

7.1.2 Security & privacy attacks

Since the questionnaires are served to the customers through e.g. a web application, we must consider security attacks targeted at such applications. A malicious customer could try to influence the model accuracy by tempering with the learning set. This could be done by e.g. flooding the web application with bogus questionnaire's answers. In this thesis we will assume that customers are trustworthy, and we will instead focus on privacy attacks, in particular those that target machine learning models.

If a machine learning model was trained using personal data, such as people's health records or identity information, then a privacy attack would aim at extracting these information to benefit the attacking party. For the scope of this thesis, we consider the insurance company to be the adversary, since they own the machine learning model. While they cannot access its content (as it is enclave-protected), it can design privacy attacks in order to gain information about the training set used by the model (i.e. the questionnaires submitted by the customers). Since it is assumed that the model is trained and run within a trusted environment, we will focus our threat model on black-box attacks (i.e. the attacker only has access to the model's API, can submit input vectors and retrieve their corresponding predictions), as shown in Figure 7.2 (adapted from [35]).

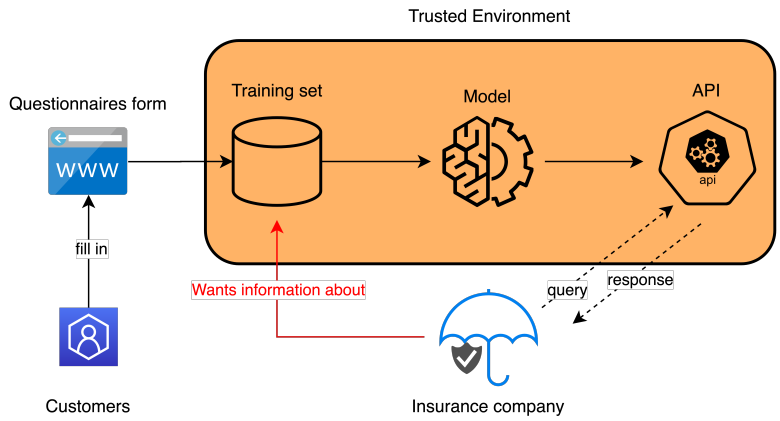


Figure 7.2: Threat model

In such a model, the insurance company is interested in gaining meaningful information about the training instances that were used by the model. Several attacks can be constructed [35]:

- **Membership inference attack**: these attacks aim at determining whether or not an input vector $\mathbf{x}$ was used as part of the training set. First introduced by Shokri et al. (2017) [37], it is one of the most popular attacks. This attack assumes a black-box scenario where the attacker only has access to the prediction vectors.
- **Model inversion attack**: given a prediction vector $\hat{\mathbf{y}}$ and partial knowledge of some features of the initial sample $\mathbf{x}$, this attack aims at recovering information about one or all missing features. This is not to be confused with *attribute inference attacks*, which try to infer sensitive feature's values of a targeted instance by leveraging publicly available data.
- **Property inference attack**: these attacks aim at extracting properties over the training set that were not explicitly encoded as features during the learning task. For instance, our synthetic data generation process (described in details in Chapter 5) does not encode the number of employees of a company in the dataset. Trying to determine such property from the model would fall under this category of attacks.

- **Model extraction attack**: these attacks do not aim at recovering information about the training dataset, but information about the inner working of the learning model, in order to reconstruct a substitute model that behaves similarly.

Many of the above attacks are conducted through *shadow models training*, which is illustrated in Figure 7.3. In shadow training, the attacker trains various models (the so-called shadow models) on shadow datasets, i.e. datasets that follow a similar distribution as the target dataset. Once the shadow models are trained, the attacker constructs an attack dataset, where each instance typically represents the probability vector outputted by the shadow models. The attacker can then train an attack model, which takes as input a prediction vector and outputs membership / property information.

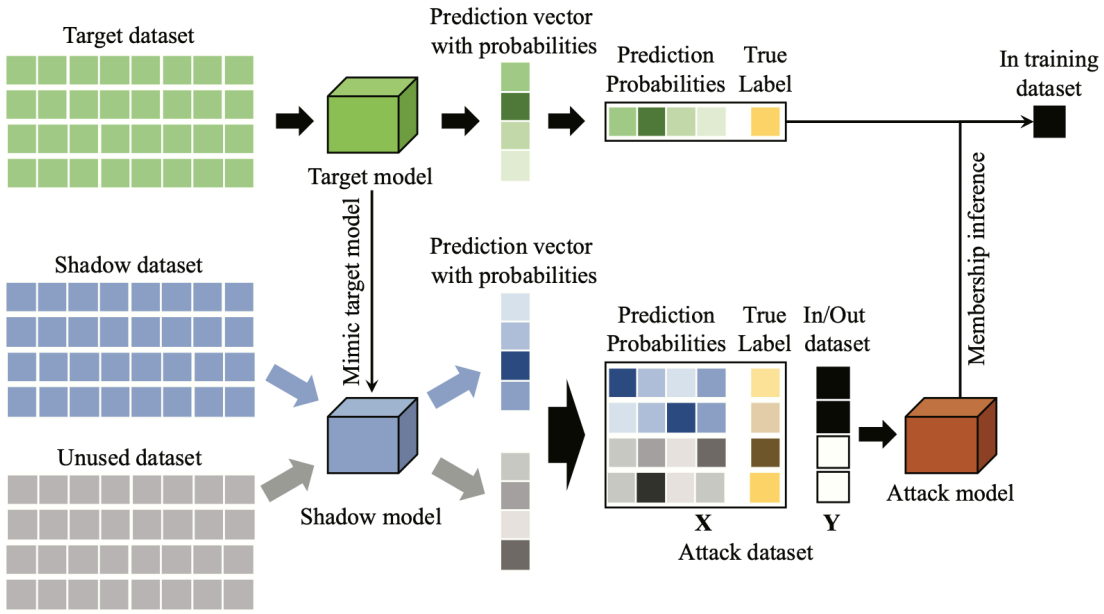


Figure 7.3: Shadow training architecture for a membership inference attack. [6]

7.2 Attacks on DP-GBDT

While there are many Python libraries readily available to conduct privacy attacks on machine learning models, such as the *adversarial-robustness-toolbox* Python library[1], to the best of our knowledge there is no work in the literature that considers privacy attacks on regression models such as our ensemble of gradient boosted decision trees. We therefore tried to convert our synthetic datasets to a classification task, and to tweak our model to support multi-class classification. We evaluate several classical attacks, in particular the gap attack, introduced by Shokri et al. (2017) [37], for different train-test split ratios. Unfortunately, our tests indicate that the current attacks do not adapt well to regression models. In particular, we were unable to detect an explainable correlation between the privacy budget and the accuracy of the membership inference attack. For reference, our results are shown in Figure A.4 and summarised in Table A.4 (for a fixed $\varepsilon = 0.1$).

While for low ε values, our model is effectively reducing the success rate of membership inference attacks to 60% or below, the literature ([6], [8]) shows that we should see such values starting from a higher ε value. Figure A.5 shows the same attack on a 50-50% train-test split. From the Figure, what happens is unclear: the model seems not to be leaking any

[1] https://github.com/Trusted-AI/adversarial-robustness-toolbox

data. However this should be interpreted with a grain of salt, as this could be the result of a lack of better attacks for GBDT models. Indeed, most of current work that evaluates the impact of differential privacy on membership inference attacks targets neural networks and deep learning models only (such as the work in [6] and [2]). The reader can refer to Figure 4 in [34] to get an idea of the influence of differential privacy against membership inference attacks for neural networks.

During this thesis, we have also tried to evaluate the model against other inference attacks, such as attribute inference attacks. Different settings were tested, such as growing very deep decision trees to overfit on the training data. This was done using both the *adversarial-robustness-toolbox* as well as other work found on popular open-source website GitHub. Results were hovering in the 50% attack accuracy zone, consistently across the different tools, confirming that our model could not be evaluated as-is. This shows that developing privacy attacks on regression ensemble models will be an important step in evaluating them, and we encourage future work in that direction.

Chapter 8

Related Work

In this chapter, we give a brief overview of other privacy preserving work, that use different mechanisms than differential privacy. In particular, we focus on membership privacy.

Nasr et al. (2018) [28] opt to turn the problem into a *min-max privacy game*, and design a training algorithm that both minimises the prediction loss of the model as well as the maximum gain of the best membership inference attack. By considering the strongest membership inference attack available to an attacker, the defender can ensure that his model will provide the best protection. Their method is designed to work on classification models. They showed that on popular classification dataset Purchase-100, their method only decreased baseline accuracy by 3.6%, while reducing the membership attack success rate from 67.6% to 51.6%.

In [23], Jia et al. (2019) introduced MemGuard. MemGuard offers defences against membership privacy attacks while providing utility-loss guarantees. As opposed to tampering with the training process of the algorithm such as in differential privacy, the authors propose to carefully add noise to the vector of confidence score. This vector is equivalent to the prediction vector with probabilities that is depicted in Figure 7.3. The amount of noise that is added to the prediction vector is not enough to change the predicted label itself, but is enough to trick the adversary's attacking model.

In their design of a membership inference attack, Shokri et al. (2017) [37] also give pointers to mitigation strategies:

1. **Restrict the prediction vector to the top k classes only**: the more classes there are in a dataset, the more information the model leaks. If there are many classes that are present in small quantities only, restricting the model to the top k classes will still result in useful outputs. Having a small k will reduce the attack success.
2. **Coarsen precision of the prediction vector**: this is similar to [23], but instead of adding noise to the prediction vector, its probabilities are rounded to d digits. The smaller d is, the less information the model leaks.
3. **Use regularization**: since models that overfit too much the data will tend to be more vulnerable to membership inference attacks, regularisation techniques such as the L_2-norm standard regularisation can be used to counter overfitting.

As shown in above research, differential privacy is not the one solution to many problems, but rather it is a potential solution to some specific problems, and sometimes differential privacy is not the way to go. Membership privacy (or more generally privacy preserving machine learning) is an active area of research, and there's still room for improvement and for finding a one-fit-all solution.

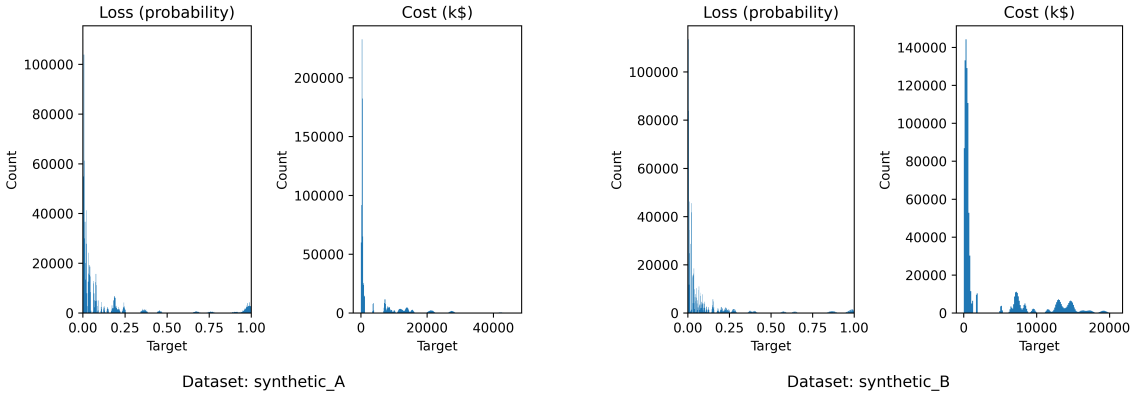
Chapter 9

Conclusion

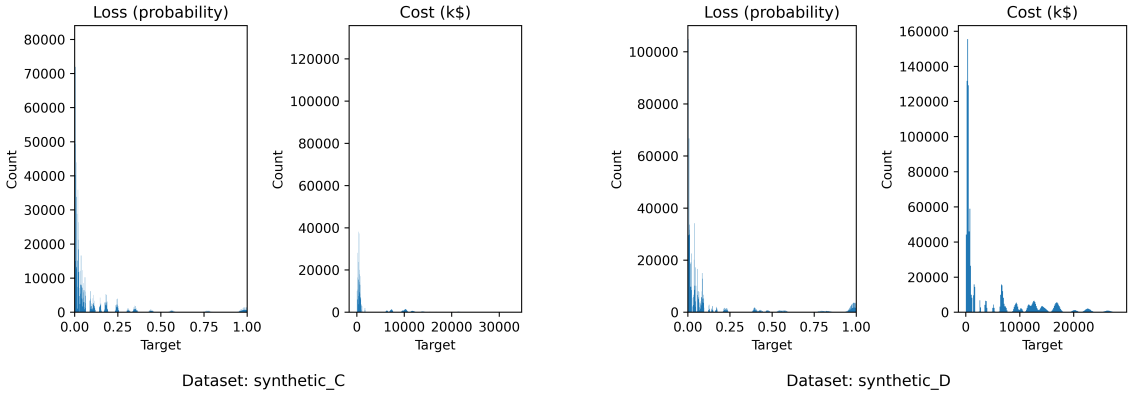
In this work, we merge existing techniques for inducing differentially private decision trees and additionally propose a new induction method, *2-nodes*, which enhances accuracy over both real and synthetic datasets. We show that under strong privacy constraints, our model achieves great accuracy on our synthetic datasets. While this work shows encouraging results in assessing cyber risk, there are still several paths to be explored in future work. First, while the model currently supports both regression and classification tasks, finding optimal privacy bounds (tighter sensitivity bounds lead to enhanced accuracy) for multi-class classification remains an open challenge. Second, and as seen during our model's evaluation, the current literature does not cover privacy leakage attacks targeted at regression models, much less ensemble models such as our gradient boosted decision trees. Developing such attacks would be crucial to properly evaluate our work, and thus constitute a challenging and interesting future research direction. Finally, the space of ensemble methods and their respective induction algorithms is yet another area of potential improvement, that could contribute towards better privacy preserving machine learning methods.

Appendix A

Appendix

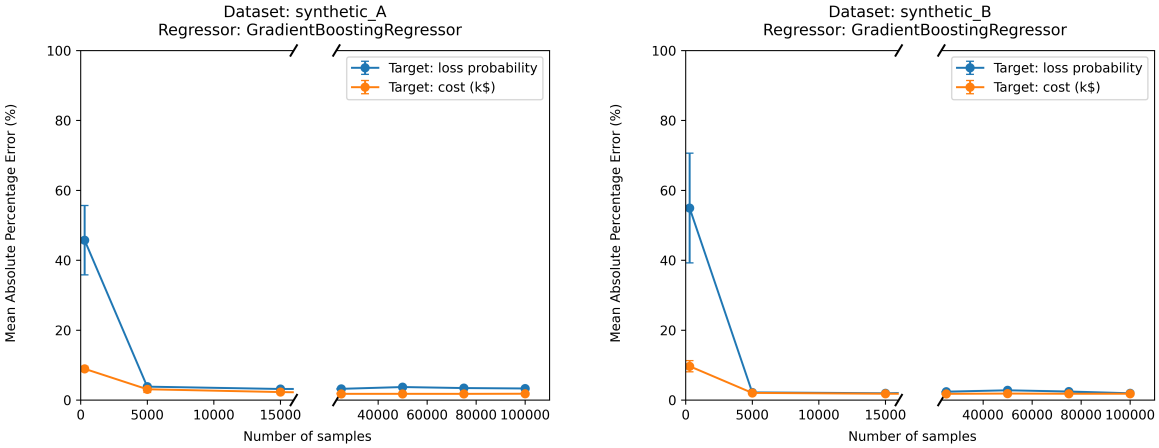


(a) Synthetic datasets A and B

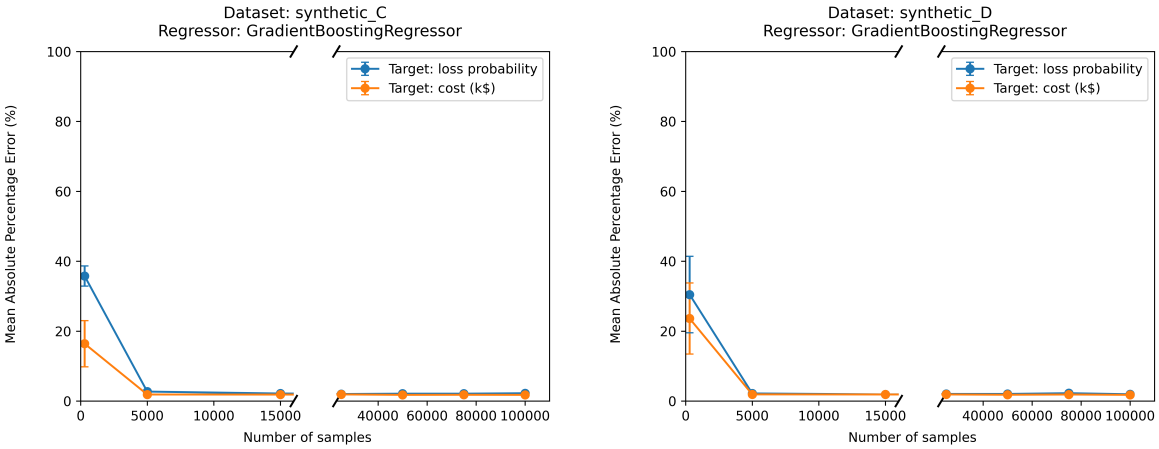


(b) Synthetic datasets C and D

Figure A.1: Target distribution for the synthetic datasets.



(a) Synthetic datasets A and B



(b) Synthetic datasets C and D

Figure A.2: Problem difficulty for the synthetic datasets.

Feature	Meaning
company_size	One of [*small*, *large*]. Refers to the size of the company.
company_sector	The sector the company operates in. Refer to Table 5.2 for the list of sectors.
company_allows_BYOD	The company has a *bring your own device* policy, which allows employees to use their personal devices as their work device.
company_does_pentest	The company runs penetration tests against their infrastructure, at least once a year.
company_uses_RBAC	The company has a *role based access control* policy in place.
company_does_red_teaming	The company performs red teaming exercises.
company_has_2FA	The company uses 2-factor authentication.
company_has_password_policy	The company has a password policy in place. Such policy defines e.g. the minimum length of employee passwords.
company_trains_employees_against_social_engineering	The company trains its employees against social engineering techniques. This includes training against e.g. phishing.
company_has_antivirus	The company has antivirus in place.
company_has_patching_process	The company keeps its systems up to date by applying the latest software / security patches on a regular basis.
company_has_ingress_firewall	The company has ingress fire-walling rules.
company_has_egress_firewall	The company has egress fire-walling rules.
company_uses_3rd_party	The company relies on a 3rd party provider for some of its services. This can be e.g. cloud-based solutions.
company_has_ciso	The company has a dedicated CISO.
company_has_threat_intelligence_team	The company has a dedicated threat intelligence team.
company_monitors_IOCs	The company is subscribed to an indicator of compromise feed.
company_has_IDS	The company has an intrusion detection system.
company_has_IPS	The company has an intrusion prevention system.
company_separates_systems	The company separates its systems, e.g. they have a DMZ.
company_does_daily_backup	The company performs daily backups of its systems.
company_has_recovery_plan	In the event of a security incident or other disaster, the company has a process for recovery in place. This can include people to contact, processes to start, etc.
company_has_incident_response_team	The company has a dedicated IR team.

Table A.1: List of features and their meaning for the synthetic datasets.

Feature (F)	$P(F)$	$P(F\|loss)$
company_size	small: 0.68 large: 0.32	
company_sector	See Table 5.1	
company_allows_BYOD	0.44	0.14
company_does_pentest	0.36	0.23
company_uses_RBAC	0.85	0.67
company_does_red_teaming	0.33	
company_has_2FA	0.78	
company_has_password_policy	0.88	
company_trains_employees_against_social_engineering	0.70	
company_has_antivirus	0.95	0.81
company_has_patching_process	0.60	0.828
company_has_ingress_firewall	0.83	0.84
company_has_egress_firewall	0.61	0.76
company_uses_3rd_party	0.87	0.79
company_has_ciso	0.67	0.68
company_has_threat_intelligence_team	0.38	
company_monitors_IOCs	0.44	
company_has_IDS	0.94	
company_has_IPS	0.81	
company_separates_systems	0.79	
company_does_daily_backup	0.91	
company_has_recovery_plan	0.33	
company_has_incident_response_team	0.46	

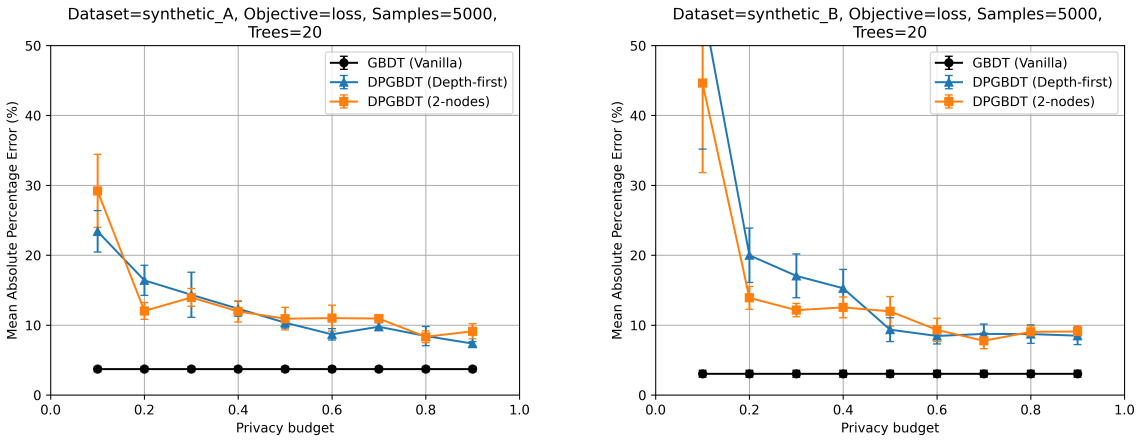
Table A.2: List of probabilities used in the Bayesian Network in Figure 5.6. Conditional probabilities that rely on multiple features are omitted but are available in the implementation.

	(Non-DP) Vanilla	(DP) Depth-first	(DP) 2-nodes
Synthetic A (MAPE)	3.71 ± 0.37	$\mathbf{10.35 \pm 0.76}$	10.92 ± 1.61
Synthetic B (MAPE)	3.05 ± 0.49	$\mathbf{9.35 \pm 1.69}$	11.97 ± 2.11
Synthetic C (MAPE)	2.33 ± 0.23	9.25 ± 0.56	$\mathbf{9.24 \pm 1.13}$
Synthetic D (MAPE)	3.00 ± 0.83	$\mathbf{13.08 \pm 1.54}$	16.06 ± 4.06

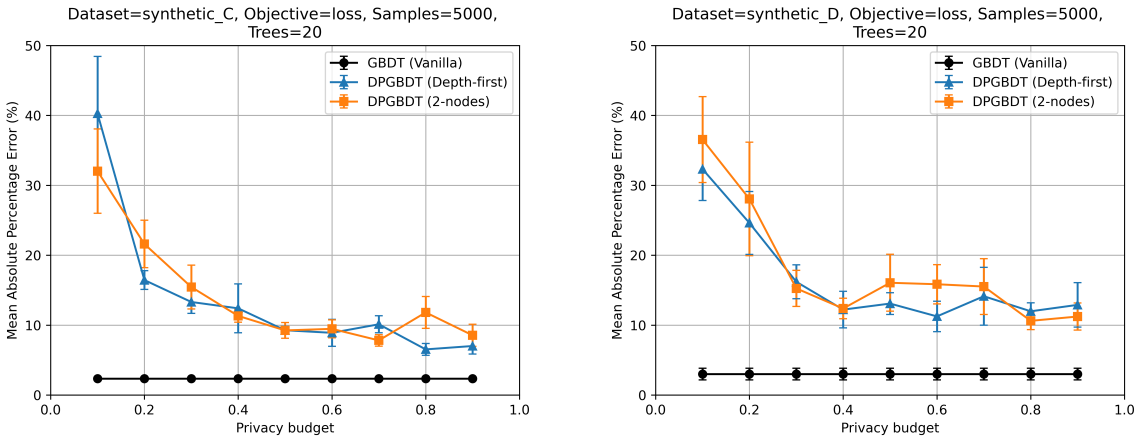
Table A.3: Prediction error for the *loss* target on the synthetic datasets.

	(Non-DP) Vanilla	(DP) Depth-first	(DP) 2-nodes
Synthetic A	74.06	**41.87**	44.89
Synthetic B	74.02	47.77	**47.31**
Synthetic C	73.48	**45.36**	51.08
Synthetic D	73.70	51.16	**47.89**

Table A.4: Membership inference attack success rate (in %) for the *cost* target on the synthetic datasets. Train-test split is set to 75-25%.

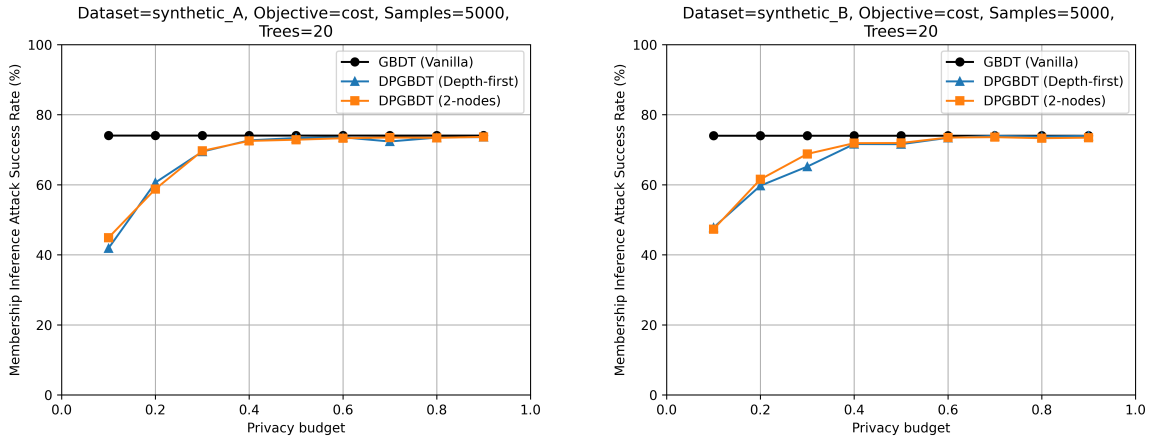


(a) Synthetic datasets A and B

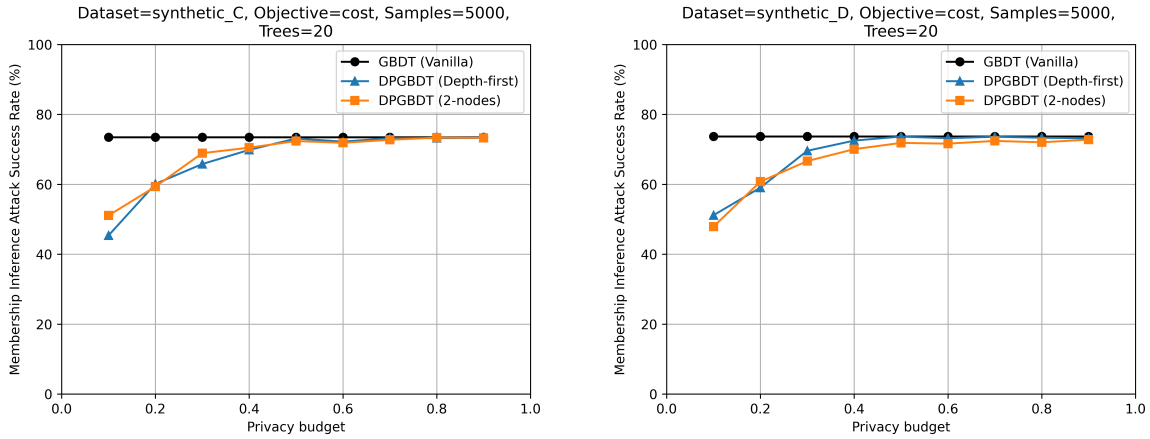


(b) Synthetic datasets C and D

Figure A.3: Mean Absolute Percentage Error for the *loss* target on the synthetic datasets.

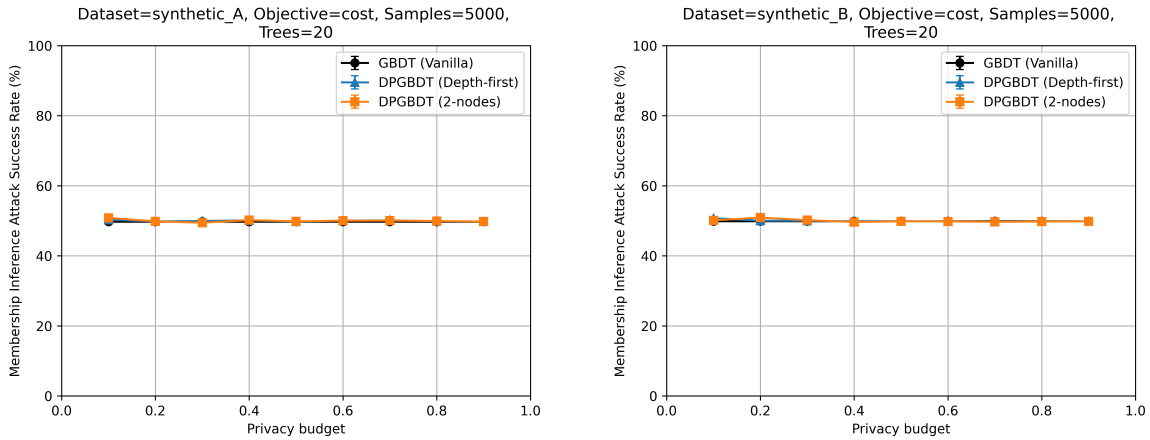


(a) Synthetic datasets A and B



(b) Synthetic datasets C and D

Figure A.4: Membership inference attack success rate (in %) for the *cost* target on the synthetic datasets. Train-test split is set to 75-25%.



(a) Synthetic datasets A and B

Figure A.5: Membership inference attack success rate (in %) for the *cost* target on the synthetic datasets. Train-test split is set to 50-50%.

Bibliography

[1] AIG. *Claims Intelligence Series*.

[2] Daniel Bernau, Philip-William Grassal, Jonas Robl, and Florian Kerschbaum. Assessing differentially private deep learning with membership inference. *CoRR*, abs/1912.11328, 2019.

[3] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: the sulq framework. In *PODS*, pages 128–138. ACM, 2005.

[4] Nazanin Borhan. *Budget Allocation on Differentially Private Decision Trees and Random Forests*, 2018.

[5] Checkpoint. *Cyber Security Report*, 2020.

[6] Junjie Chen, Wendy Hui Wang, and Xinghua Shi. Differential privacy protection against membership inference attack on machine learning for genomic data. *bioRxiv*, 2020.

[7] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *KDD*, pages 785–794. ACM, 2016.

[8] Christopher A. Choquette-Choo, Florian Tramèr, Nicholas Carlini, and Nicolas Papernot. Label-only membership inference attacks. *CoRR*, abs/2007.14321, 2020.

[9] Cisco. *Annual Cybersecurity Report*, 2018.

[10] Cynthia Dwork. Differential privacy. In *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 338–340. Springer, 2011.

[11] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[12] Wei Fan, Haixun Wang, Philip S. Yu, and Sheng Ma. Is random model better? on its accuracy and efficiency. In *ICDM*, pages 51–58. IEEE Computer Society, 2003.

[13] Sam Fletcher and Md Zahidul Islam. A differentially private decision forest. In *AusDM*, volume 168 of *CRPIT*, pages 99–108. Australian Computer Society, 2015.

[14] Sam Fletcher and Md Zahidul Islam. A differentially private random decision forest using reliable signal-to-noise ratios. In *Australasian Conference on Artificial Intelligence*, volume 9457 of *Lecture Notes in Computer Science*, pages 192–203. Springer, 2015.

[15] Sam Fletcher and Md Zahidul Islam. Differentially private random decision forests using smooth sensitivity. *Expert Syst. Appl.*, 78:16–31, 2017.

[16] Sam Fletcher and Md Zahidul Islam. Decision tree classification with differential privacy: A survey. *ACM Comput. Surv.*, 52(4):83:1–83:33, 2019.

[17] Arik Friedman and Assaf Schuster. Data mining with differential privacy. In *KDD*, pages 493–502. ACM, 2010.

[18] Abigail Goldsteen, Gilad Ezov, Ron Shmelkin, Micha Moffie, and Ariel Farkash. Anonymizing machine learning models. *CoRR*, abs/2007.13086, 2020.

[19] Xiaofei (Rex) Guo Gorka Irazoqui. *LLC Attacks - Applicability & Countermeasures*.

[20] Aurélien Havet, Rafael Pires, Pascal Felber, Marcelo Pasin, Romain Rouvoy, and Valerio Schiavoni. Securestreams: A reactive middleware framework for secure data stream processing. *CoRR*, 2018.

[21] IBM. *X-Force Threat Intelligence Index*, 2020.

[22] Geetha Jagannathan, Krishnan Pillaipakkamnatt, and Rebecca N. Wright. A practical differentially private random decision tree classifier. In *ICDM Workshops*, pages 114–121. IEEE Computer Society, 2009.

[23] Jinyuan Jia, Ahmed Salem, Michael Backes, Yang Zhang, and Neil Zhenqiang Gong. Memguard: Defending against black-box membership inference attacks via adversarial examples. In *CCS*, pages 259–274. ACM, 2019.

[24] Michael J. Kearns and Yishay Mansour. On the boosting ability of top-down decision tree learning algorithms. *J. Comput. Syst. Sci.*, 58(1):109–128, 1999.

[25] Qinbin Li, Zhaomin Wu, Zeyi Wen, and Bingsheng He. Privacy-preserving gradient boosting decision trees. In *AAAI*, pages 784–791. AAAI Press, 2020.

[26] Xiaoqian Liu, Qianmu Li, Tao Li, and Dong Chen. Differentially private classification with decision tree ensemble. *Appl. Soft Comput.*, 62:807–816, 2018.

[27] Noman Mohammed, Rui Chen, Benjamin C. M. Fung, and Philip S. Yu. Differentially private data release for data mining. In *KDD*, pages 493–501. ACM, 2011.

[28] Milad Nasr, Reza Shokri, and Amir Houmansadr. Machine learning with membership privacy using adversarial regularization. *CoRR*, 2018.

[29] NetDiligence. *Cyber Claims Study*, 2019.

[30] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. A survey of published attacks on intel SGX. *CoRR*, abs/2006.13598, 2020.

[31] Kobbi Nissim, Sofya Raskhodnikova, and Adam D. Smith. Smooth sensitivity and sampling in private data analysis. In *STOC*, pages 75–84. ACM, 2007.

[32] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

[33] J. Ross Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, 1986.

[34] Md. Atiqur Rahman, Tanzila Rahman, Robert Laganière, and Noman Mohammed. Membership inference attack against differentially private deep learning model. *Trans. Data Priv.*, 11(1):61–79, 2018.

[35] Maria Rigaki and Sebastian Garcia. A survey of privacy attacks in machine learning. *CoRR*, abs/2007.07646, 2020.

[36] Haijian Shi. Best-first decision tree learning. 2007.

[37] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy*, pages 3–18. IEEE Computer Society, 2017.

[38] Si Si, Huan Zhang, S. Sathiya Keerthi, Dhruv Mahajan, Inderjit S. Dhillon, and Cho-Jui Hsieh. Gradient boosted decision trees for high dimensional sparse output. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 3182–3190. PMLR, 2017.

[39] Todd A. Stephenson. *An Introduction To Bayesian Network Theory and Usage*, 2000.

[40] Verizon. *Data Breach Investigations Report*, 2020.

[41] Kaiwen Wang, Travis Dick, and Maria-Florina Balcan. Scalable and provably accurate algorithms for differentially private distributed decision tree learning. *CoRR*, abs/2012.10602, 2020.

[42] Katherine Woodruff. *Introduction to boosted decision trees*, 2017.

[43] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *USENIX Security Symposium*, pages 719–732. USENIX Association, 2014.

[44] Lingchen Zhao, Lihao Ni, Shengshan Hu, Yanjiao Chen, Pan Zhou, Fu Xiao, and Libing Wu. Inprivate digging: Enabling tree-based distributed data mining with differential privacy. In *INFOCOM*, pages 2087–2095. IEEE, 2018.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

PRIVACY-PRESERVING MACHINE LEARNING FOR CYBER INSURANCE

Authored by (in block letters):

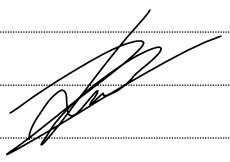
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
GIOVANNA	THEO

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
ZURICH, 23/02/2021	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.